CASEVision™/ClearCase Administration Guide

Document Number 007-1774-020

CONTRIBUTORS

Written by John Posner
Engineering contributions by Atria Software, Inc., Lisa Kvarda, and Trevor Bechtel.

CASEVision™/ClearCase Administration Guide
Document Number 007-1774-020

Contents

1. Administrator's View of a ClearCase Network 1
Network Overview 1
ClearCase Hosts 2
ClearCase Data Storage 3
Versioned Object Bases (VOBs) 3
Views 4
ClearCase User Base 5
Registries for VOBs and Views 6
Network Regions 6
ClearCase Client-Server Processing 8
ClearCase Servers 8
Server Error Logs 9
ClearCase Startup and Shutdown 10

2. ClearCase Data Storage 11
Versioned Object Bases (VOBs) 11
VOB Database 12
VOB Storage Pools 13
Source Storage Pools 13
Cleartext Storage Pools 14
Derived Object Storage Pools 14
The *vob_server* Process 15
Default, Local, and Remote Storage Pools 15
Elements' Source Pool Assignments 17
Commands for Working with Storage Pools 17
Views 18
View Database 19
View's Private Storage Area 20

3. Network-Wide Access to ClearCase Data 23
Storage Directories and Access Paths 23
Distributed VOBs and Views 23
Storage Registries 24
Object Registries 24
Tag Registries 25
Network-Wide Accessibility of VOBs and Views 27
Public and Private VOBs 27
Network Regions 28
Registries in a Multiple-Region Network 30
Tag Registry Implementation 31
Establishing Network Regions 33
Recording Multiple Network Interfaces 33
ClearCase Data and Non-ClearCase Hosts 34
Usage Restrictions 34
Building on a Non-ClearCase Host 34
Using *automount* with ClearCase 35
Use of *–hosts* Map Required 35
Specifying a Non-Standard Mount Directory 35

4. User-Level Access to ClearCase Data 37
Users: Usernames and Groups 37
Network-Wide ClearCase Administrator 39
VOBs and Views: Owner and Groups 39
The .identity Directory 40
Effect of Creator's 'umask' Setting 42
Example: Two VOBs and Two User Groups 42
Creating the VOBs 43
Controlling Access to the VOBs 45

Access to Individual File System Objects 45
Access-Control Settings 45
Notes on VOB/View Interactions 46
Initialization of Access-Control Settings for VOB Objects 47
Initialization of Access-Control Settings for View Objects 47
Access-Control Settings for Physical Data Storage 47
How Processes Access ClearCase Data 48
Read Access by Processes 51
Write Access by Processes 51
ClearCase-Level Access Permissions 52
Locks on VOB Objects 53
Locking Type Objects 54

5. **ClearCase User Licensing Scheme** 55
Floating License Architecture 55

6. **Setting Up ClearCase VOBs** 57
Selecting a VOB Host 57
Planning for One or More VOBs 58
Planning for Release VOBs 60
Modifying a VOB Host for ClearCase 61
Kernel Resources 61
Optional Software Packages 61
Creating a New VOB 62
Adjusting the VOB's Identity Information 63
Case 1: One Group for All VOBs, Views, and Users 64
Case 2: Accommodating Multiple User Groups 64
Example: Multiple Groups 64
Ensuring the VOB's Global Accessibility 65
Case 1: Heuristic Guess Was Right 66
Case 2: Guess Was Wrong, But Global Pathname Does Exist 66
Case 3: Global Pathname Does Not Exist 66
Creating Remote Storage Pools 67
Coordinating the New VOB with Existing VOBs 68

Populating a VOB with Data 68
Example: Importing RCS Data 69
Creating the Conversion Scripts 69
Running the Conversion Scripts 70

7. Setting Up ClearCase Views 71
Setting Up an Individual User's View 71
View Storage Requirements 72
View Database 72
View's Private Storage Area 72
Setting Up a Shared View 73
Setting Up an Export View for Non-ClearCase Access 74
Exporting Multiple VOBs 75
Multihop Export Configurations 76
Restricting Exports to Particular Hosts 78

8. Preventing Accidental Deletion of Data by *crontab* Entries 79
Preventing Recursive Traversal of '/' 79
Crontab Modification During ClearCase Installation 80
Modifying a Crontab Entry 80
Preventing Accidental Deletion of the Lock Manager Socket 82

9. Data Backup: VOBs and Views 83
Backup Tools 83
Backing Up a VOB 83
Determining a VOB's Location 84
Ensuring a Consistent Backup 84
Partial Backups 85
Example of Partial VOB Backup 86
Backing Up a VOB with Remote Storage Pools 86
Restoring a VOB from Backup 87
Reestablishing Consistency of a View's "Derived Object State" 90
Backing Up a View 91
Restoring a View from Backup 92

10. Periodic Maintenance of the Data Repository 95
VOB Storage Maintenance 95
Scrubbing VOB Storage Pools 96
Scrubbing VOB Databases 97
Database Scrubbing: Logical vs. Physical 97
View Storage Maintenance 98
Scrubbing View-Private Storage 99
User-Supplied Maintenance Procedures 99
Caution: 'Local' Scripts May Not Really be Local 100

11. Occasional VOB Maintenance 101
Moving a VOB (Same Architecture) 101
Moving a VOB (Different Architecture) 104
Removing Unneeded Versions from a VOB 108
Example 109
Restoring a Single Element From Backup 110
Creating Additional VOB Storage Pools 113
Caution on Remote Source Pools 114
Example: Assigning All Files in a Directory to a New Pool 115
Example: Moving an Existing Storage Pool to Another Disk 116
Adjusting Storage Pool Scrubbing 117
Scrubbing Derived Objects More Often 117
Fine-Tuning Derived Object Scrubbing 118
Scrubbing Less Aggressively 120

12. Occasional View Maintenance 121
Moving a View (Same Architecture) 121
Moving a View (Different Architecture) 123
Moving a View's Private Storage Area 126
Manual Cleanup of a View 127

13. ClearCase Performance Tuning 131
Improving VOB Host Performance 131
Eliminate Extraneous Processes 131
Manipulate Block Buffer Caches 132
Block Buffer Cache Statistics 133
Flushing of the Block Buffer Cache 133
Improving Client Host Performance 133
Increasing System Resources 133
Creating Remote Storage Pools 134
Caution on Remote Source Pools 134
Changing the MVFS Configuration (SunOS Only) 135
Selecting Alternative Cache Size Defaults—SunOS 4 Only 136
Compiling New Cache Sizes into the MVFS 137
SunOS 4 Cache Override Procedure 138
SunOS 5 Cache Override Procedure 139
Reconfiguring a View 140

14. Making a VOB or View Inaccessible 141
Alternative to VOB Deactivation 141
Taking a VOB Out of Service 141
Restoring the VOB to Service 143
Taking a View Out of Service 144
Restoring the View to Service 144
Permanent Removal of a VOB or View 144

15. Determining a Data Container's Location 145
Scenario 145
Determining the ClearCase Status of Files 145
Determining the Full UNIX Pathnames of Files 146
Where is the VOB? 146
Where is the View? 147

Where are the Individual Files? 147
Locating a Checked-Out Version 148
Locating a Checked-In Version's Cleartext Container 148
Locating a Checked-In Version's Source Container 149
Locating a View-Private File 149
Non-Local Storage 149
Links and Directories 150

16. Adjusting the ClearCase Startup/Shutdown Script 151
Name of Startup/Shutdown Script 151
Changing VOB Mounts to "Release 1 Style" 151
Changing Dynamic Loading of the MVFS 153

17. Adjusting ClearCase License Information 155

18. Adjusting ClearCase Registry Information 157
Registry Review 157
ClearCase Storage Registry 157
Object Registry Files 158
Tag Registry Files 158
cleartool Commands and Registry Files 159
lsvob and *lsview (-long)* 159
mkvob and *mkview* 160
mktag and *rmtag* 161
Adding a Network Region 162
When to Partition the Network 163
A Sample Network Partition 163
Procedure for Adding a Network Region 165
Moving a Host to a New Network Region 168
Removing a Network Region 169
Registry-Related Guidelines 169
Multiple Network Regions 171

19. Changing the Location of Network-Wide Resources 173
Changing the Location of the Release Area 173
Renaming the Release Host 173
Moving Licenses to Another Host and Renaming a License Server Host 174
Moving the ClearCase Registry 174
Renaming the Registry Server Host 175
Renaming a ClearCase Host 175

Index 177

Figures

Figure 1-1 ClearCase Storage Registries 7
Figure 1-2 Client-Server Processing 9
Figure 2-1 VOB Database and VOB Storage Pools 13
Figure 2-2 Local and Remote VOBStorage Pools 16
Figure 3-1 ClearCase Object and Tag Registries (Single Network Region) 26
Figure 3-2 Network with Global Naming 28
Figure 3-3 Network Regions and Their Tag Registries 32
Figure 4-1 Establishing a User's Group Assignments 38
Figure 4-2 Multiple-Group Support for VOBs and Views 40
Figure 4-3 The .identity Directory of a VOB or View 41
Figure 4-4 Planning Access to Development Sources 43
Figure 4-5 Data Access Paths 50
Figure 4-6 Read' Access through Group Membership 51
Figure 4-7 'Write' Access through Group Membership 52
Figure 6-1 Linking Multiple VOBs Into a Single Directory Tree 60
Figure 7-1 Export View for Non-ClearCase Access 75
Figure 7-2 Avoiding Access Cycles in Non-ClearCase Access 77
Figure 8-1 Viewroot Directory as a Super-Root 80
Figure 10-1 Controlling VOB Growth 96
Figure 18-1 ClearCase Storage Registry 158
Figure 18-2 *cleartool* Commands and the ClearCase Storage Registry 162
Figure 18-3 Sample Network with Two Regions 164

Tables

Table 9-1 VOB Components for Partial Backups 85

Table 13-1 Selecting the Default or Alternative MVFS Cache Configuration 135

Table 13-2 Cache Parameters for MVFS module: ‘mvfs.o’ 137

Table 15-1 Storage Locations of MVFS Files 147

Chapter 1

Administrator's View of a ClearCase Network

This chapter presents a system administrator's overview of a local area network using ClearCase. It also serves as a roadmap to other chapters in this manual, and to detailed reference information in the *CASEVision™/ClearCase Reference Pages*.

Network Overview

A local area network using ClearCase includes these principal components:

- **Hosts**—ClearCase can be installed and used on any number of hosts in a network. Different hosts use ClearCase software in different ways; for example, one host might be used only to store version-controlled data; another might be used only to run ClearCase software development tools.
- **Data storage**—ClearCase data is stored in *VOBs* and *views*, which can be located on any or all the hosts where ClearCase is installed. A VOB or view can have auxiliary data storage on hosts where ClearCase is *not* installed; such storage is accessed through standard symbolic links.

  For many organizations, the set of all VOBs constitutes a *central data repository*, which you may need to administer as a unit. Most views are used by individuals; it is likely, however, that one or more shared views will be created, requiring some central administration.
- **User base**—ClearCase is used by a set of people, each of whom has a *username* and is assigned to one or more *groups*. Any number of people can use ClearCase on any number of hosts; the licensing scheme limits the number of *concurrent* users, but does not limit the number of hosts.

ClearCase Hosts

ClearCase is a distributed application with a client-server architecture. This means that any particular development task (for example, execution of a single ClearCase command) may involve programs and data on several hosts. It is important to classify hosts by the roles they play, because different kinds of hosts require different administrative procedures. But keep in mind that any particular host may play different roles at different times, or several roles at once.

- Network-wide release host—One host in the network acts as the network-wide release host. This host stores the entire ClearCase release, exactly as it is supplied on the distribution medium (magnetic tape, CD-ROM). Note that this release area is active storage, not archival storage—some individual developers' workstations may access ClearCase programs and/or data through symbolic links to the release area.
- License server host(s)—One or more hosts in the network act as ClearCase license server hosts, authorizing and limiting ClearCase usage according to the terms of your license agreement. Each host on which ClearCase is installed is assigned to a particular license server host, and periodically communicates with that host. (The albd_server process on a license server host acts as the "license server process".)
- Registry server host—One host in the network acts as the ClearCase registry server host. This host stores a set of files that contain essential access-path information concerning all the VOBs and views in the network. ClearCase client and server programs on all other hosts occasionally communicate with the registry server host, in order to determine the actual storage location of ClearCase data. (The albd_server process on the registry server host acts as the "registry server process".)
- Client hosts—Each user typically has his or her own workstation. It is called a client host, because it runs ClearCase client programs: the programs installed in /usr/atria/bin, including cleartool, clearmake, and xclearcase.

  ClearCase must be explicitly installed on each client host; installation must include the multiversion file system (MVFS), a ClearCase virtual file system extension. All access by client programs to ClearCase data (in VOBs and views) goes through the host's MVFS.

- **Server hosts**—Some hosts may be used only as data repositories for VOB storage directories and/or view storage directories. Such *server hosts* run ClearCase *server programs* only: *albd_server*, *vob_server*, *view_server*, and other programs installed in */usr/atria/etc*.

  ClearCase must be explicitly installed on each server host; installation need not include the MVFS—it is required only for running client programs.

- **Non-ClearCase hosts**—ClearCase need not be installed on every host in your network. In fact, it may not even be possible to install it on some hosts—those whose architectures are not (yet) supported by ClearCase. Such hosts cannot run ClearCase programs, but they *can* access ClearCase data, through standard UNIX network file system facilities. You administer these "export" (or "share") mechanisms using standard UNIX tools.

ClearCase Data Storage

All ClearCase data is stored in VOBs and views. These data structures can be distributed throughout the local area network—even an individual VOB or view can be distributed. Users see these structures as global resources; after a VOB or view is explicitly *activated,* users access it through its *VOB-tag* or *view-tag*.

Versioned Object Bases (VOBs)

The network's permanent data repository is conceptually a centralized resource. Typically, however, the repository is distributed among multiple *versioned object bases* (*VOBs*), located on multiple hosts. Each VOB is implemented as a *VOB storage directory* (actually a directory tree), which holds both developers' file system objects and an embedded database.

Administration of a network's VOBs includes:

- **Registration**—All VOBs are listed in a network-wide storage registry. In a typical network, registry maintenance is minimal—certain ClearCase commands update the registry automatically. You'll need to adjust the registry if you move a VOB to another location (for example,

to another host). You'll also need to do some registry work if different pathnames must be used on different hosts to access the same VOBs.

- **Backup**—As your organization's "family jewels", VOBs should be backed up frequently and reliably. ClearCase does not include data-backup tools; use system-supplied or third-party tools.
- **Periodic maintenance**—Administering the central repository requires continual balancing of the need to preserve important data with the need to conserve disk space. ClearCase includes tools for occasional *scrubbing* of unneeded data. You can control the meaning of "unneeded" on a per-VOB basis.

  ClearCase installation automatically sets up *crontab*(1) scripts for a host's *root* user. By default, the scripts perform daily and weekly scrubbing of all VOBs on that host. You can fine-tune VOB maintenance by revising scrubbing parameters, by revising the scripts themselves, or by adding your own scripts.
- **Access control**—Each VOB has a *principal group* and a supplementary *group list*. Together, these control which developers can use the VOB. As your organizational structure changes (for example, a new project is launched), you may need to adjust a VOB's group list.
- **Growth**—As new projects are launched (or existing projects are brought under ClearCase control), you'll need to create new VOBs, define their accessibility to various groups, and incorporate them into your data backup and periodic maintenance schemes.
- **Reformatting**—When you install a new major release of ClearCase, it may be necessary to *reformat* your existing VOBs. This process updates the *schema* of the embedded VOB database.

Views

Short-term storage for data created during the development process is provided by ClearCase *views*. A view stores checked-out versions of file elements, *view-private* files that have no counterpart in the VOB (for example, text editor backup files), and newly-built derived objects.

Developers think of views and VOBs as being very different: briefly, a VOB is where data resides; a view is a "lens" through which a developer sees VOB data. From an administrator's standpoint, however, views and VOBs are

quite similar. Each view is implemented as a *view storage directory* (actually a directory tree), which holds both developers' file system objects and an embedded database. View administration is similar to that for VOBs, including registration, backup, periodic maintenance, and access control.

Like a VOB, a view includes both a storage area for file system data, and an associated database:

- The view's *private storage area* (subdirectory *.s* of the top-level view storage directory) holds all view-private objects. It also holds the data files (data containers) for derived objects built in the view.
- The *view database* tracks the correspondence between certain VOB database objects and view-private objects.

Most VOBs are long-lived structures, created by an administrator; views are usually created by individual developers, and tend to be shorter-lived.

ClearCase User Base

ClearCase does not, itself, maintain a registry of its users. Any user logged in to a ClearCase host can use the software, if they can acquire a ClearCase license.

Successful use of ClearCase depends on network-wide consistency in the user base: users should have the same *user-IDs* and *group-IDs* on all hosts. Consistency is usually achieved by using network-wide databases at the operating system level, such as NIS *passwd* and *group* maps.

Each user has a user-ID, a principal group-ID (specified in the OS-level password database), and a supplementary list of group-IDs (specified in the OS-level group database). These identities control the user's permission to read and create ClearCase data:

- Many *cleartool* commands check the user's identity before granting access to particular objects—element, version, and so on.
- Standard commands that access ClearCase data are also subject to access control.

Registries for VOBs and Views

All ClearCase data storage areas—all VOBs and views—in a local area network are centrally registered, on the ClearCase *registry server host*. This host has two kinds of registries:

- **Object registry**—records the physical storage locations of VOBs and views
- **Tag registry**—records the user-level access paths to VOBs and views

For example, an object registry entry might record the fact that a VOB's storage directory is located on host *neptune*, at pathname */vobstore/project.vbs*; a corresponding tag registry entry might record the fact that on each developer's workstation, the VOB is to be activated (mounted) at the location specified by its VOB-tag, */vobs/proj*.

Similarly, an object registry entry might indicate that a view storage directory is */usr/shared/integ.vws* on host *einstein*; a corresponding tag registry entry might enable developers to access the view using the view-tag *integration*.

Network Regions

In an ideal network, all hosts would access ClearCase data storage areas using exactly the same "global" pathnames. Many networks fall short of this ideal, however. To address this situation, a network can be logically partitioned into multiple *network regions*. Conceptually, each region has its own tag registry—ClearCase data structures can be accessed with different "global" pathnames in different regions. (Physically, all tag registries are implemented in a single file.)

Figure 1-1 illustrates how registries mediate access to ClearCase data structures. The administrative benefits of network-wide registries include:

- centralized control over all components of the network's distributed data repository
- independence from architecture-specific mechanisms for mounting file systems

- ability to accommodate heterogeneous networks, and networks in which hosts have multiple names and/or multiple network interfaces
- making VOBs and views globally accessible

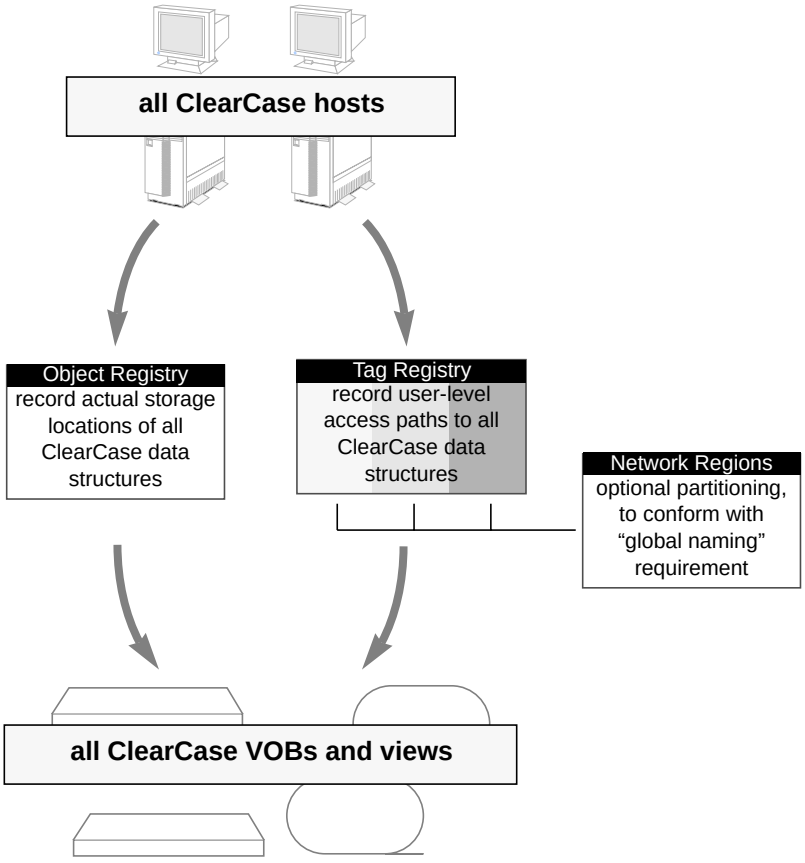


Figure 1-1 ClearCase Storage Registries

ClearCase Client-Server Processing

ClearCase is a distributed client-server application. This means that multiple processes, running on multiple hosts, can play a role in the execution of ClearCase commands—even the simplest ones. For example, when a user checks out a file element:

- The user's *client process—cleartool* or *xclearcase*—issues a "checkout" request, in the form of an RPC call.
- The RPC call is fielded by *server processes*, which run on the host where the element's VOB resides.
- A view-private copy is made of the version being checked out. This involves the *view_server* process that manages the user's view. It can also involve even more hosts:
  - The user's client process and the view may be on different hosts.
  - The view might have a private storage area that is located on a different host from the view storage directory.
  - The VOB storage pool that holds the version being checked out may be located on a different host from the VOB storage directory.

Fortunately, this is all handled automatically and reliably by the ClearCase server processes. Users need not be concerned with server-level processing at all. As an administrator, your concerns in this area typically are limited to entering an occasional "stop all ClearCase processing" command on one or more hosts; this terminates all ClearCase server processes currently active on the host.

ClearCase Servers

Each ClearCase host runs a single Atria Location Broker Daemon process, *albd_server*, which is invoked by the ClearCase startup script. (See "ClearCase Startup and Shutdown" on page 10.) Other ClearCase server processes are started, as needed, by the *albd_server* processes.

Most server processes manage a particular data structure; for example, a *view_server* process manages a particular view storage directory. Such servers always run on the host where that data structure resides. This kind of ClearCase server includes:

view_server	Manages the view storage directory of a particular view
vob_server	Manages the storage pools of a particular VOB
db_server	Fields requests from one ClearCase client program, destined for one or more VOB databases on a host
vobrpc_server	Fields requests from one or more *view_server* processes, destined for a particular VOB database

Figure 1-2 shows the communications paths connecting a client process with server processes.

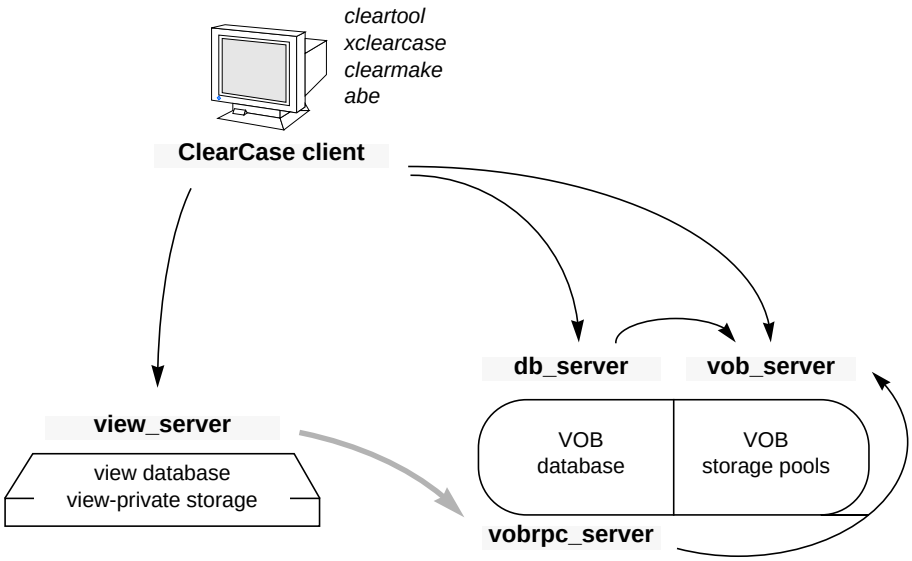


Figure 1-2 Client-Server Processing

Server Error Logs

Each ClearCase server maintains an error log on the host where it executes, in directory */usr/adm/atria/log*. Given ClearCase's distributed architecture, a user can enter a command on one host that logs an error message on another host. In such cases, the user is directed to the appropriate log file on the appropriate host.

See the *errorlogs_ccase* manual page for details on the error logs.

ClearCase Startup and Shutdown

When UNIX is bootstrapped, the ClearCase startup script is executed automatically by *init*(1M). On some systems, this script is named */etc/rc.atria*; on others, it is named */etc/init.d/atria*.The startup script:

- Starts the host's *albd_server* (Atria Location Broker Daemon) process.
- Starts a *lockmgr* process, which arbitrates concurrent access to VOB databases by multiple client processes.
- Performs additional file system setup tasks, such as mounting of VOBs.

You can also invoke this script manually, as *root*. For example:

```
# /etc/rc.atria stop
```

Stops all ClearCase processing on a host: terminates the *albd_server* and *lockmgr* processes, along with any other ClearCase server processes running on the host. User processes that are set to views on that host will also be terminated.

```
# /etc/rc.atria start
```

Restarts ClearCase processing on a host, by starting an *albd_server* process and a *lockmgr* process.

See Chapter 16, "Adjusting the ClearCase Startup/Shutdown Script", and the *init_ccase* manual page for more on this topic.

Chapter 2

ClearCase Data Storage

This chapter describes the on-disk data structures that implement ClearCase *VOBs* and *views*. File system objects stored in these structures are termed *MVFS objects*, because client programs access them through the ClearCase *multiversion file system*.

Versioned Object Bases (VOBs)

The ClearCase data repository for a network is implemented as a set of *versioned object bases* (*VOBs*). Each VOB is implemented as a UNIX directory tree, whose top-level directory is termed the *VOB storage directory.* The main components of this directory tree are:

- **VOB database**—The *db* subdirectory contains the binary files managed by ClearCase's embedded DBMS. Each VOB has its own database; there is no central database that encompasses all VOBs. The database stores several kinds of data:
  - version-control information: elements, their branch structures, and their versions
  - *meta-data* associated with the file system objects: version labels, attributes, and so on
  - *event records* and *configuration records*, which document ClearCase development activities
  - *type* objects, which are involved in the implementation of both the version-control structures and the meta-data

Actual file system data (for example, the contents of version 3 of file *msg.c*) is not stored in the VOB database.

- **VOB storage pools**—The *c*, *d*, and *s* subdirectories contain the VOB's *storage pools*, each of which is a standard UNIX directory. The storage

pools hold *data container* files, which store the VOB's file system data: versions of elements and shared binaries. Depending on the *element type*, the versions of an element might be stored in separate data container files, or might be combined into a single structured file that contains *deltas* (version-to-version differences).

- **Identity directory**—The *.identity* subdirectory contains files that establish the VOB's owner, its principal group, and its group list.

The *vob* manual page provides a detailed description of the contents of a VOB. (The *.identity* directory is not discussed further in this chapter—see "VOBs and Views: Owner and Groups" on page 39 for more on this topic.)

Note: Users do not directly access a VOB storage directory. Rather, they access a VOB through its *VOB-tag*, which specifies a location at which the VOB is activated (mounted) as a file system of type MVFS. Chapter 3 discusses this in detail. ♦

VOB Database

Each VOB has its own database, implemented as a set of files in the *db* subdirectory of the VOB storage directory. ClearCase server programs are invoked automatically, as needed, to access a VOB's database:

- A *db_server* process handles requests from a single ClearCase client program.
- A *vobrpc_server* process handles requests from one or more ClearCase *view_server* processes.

These server processes run on the host where the VOB storage directory physically resides. The *db* subdirectory must also be on that same host. (As explained below, storage pools can be remote.)

Caution: You cannot simply move the VOB database directory (*db*) to another host. See Chapter 11, "Occasional VOB Maintenance", for steps you can take if a VOB database threatens to fill up its disk partition.

For the most part, ClearCase servers and *crontab*(1) scripts manage VOB databases automatically. See Chapter 10, "Periodic Maintenance of the Data Repository" and Chapter 11 for more on VOB maintenance.

VOB Storage Pools

Each VOB has a set of storage pools, which hold several different kinds of data containers. Each storage pool holds data containers of one kind (Figure 2-1).

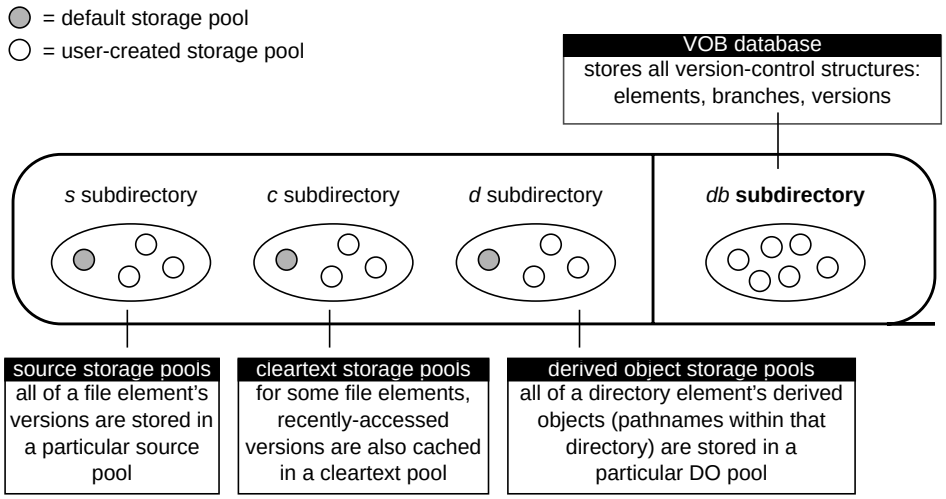


Figure 2-1 VOB Database and VOB Storage Pools

Source Storage Pools

Each *source storage pool* holds all the *source data containers* for a set of file elements. A source data container holds the contents of one or more of a file element's versions. For example, a single source data container holds *all* the versions of an element of type *text_file*. The *type manager* program for this element type handles the task of reconstructing individual versions from *deltas* in the data container. Likewise, the type manager updates the data container when a new version is checked in.

Source pools are accessed by *checkout* and *checkin* commands, and by development operations (for example, *cat*(1), *lp*(1), *cc*(1)) that read the contents of elements that are not checked-out. In many cases, however, a cleartext pool is accessed instead of the source pool.

Cleartext Storage Pools

Each *cleartext storage pool* holds all the *cleartext data containers* for a set of file elements. A cleartext data container holds the contents of one version of an element. These pools are caches that accelerate access to elements for which all versions are stored in a single data container: compressed files and text files.

For example, the first time a version of a *text_file* element is required, the *text_file_delta* type manager reconstructs the version from the element's source data container. The version is cached as a cleartext data container—an ordinary text file—located in a cleartext storage pool. On subsequent accesses, ClearCase looks first in the cleartext pool. A "cache hit" eliminates the need to access a source pool, thus reducing the load on that pool; it also eliminates the need for the type manager to reconstruct the requested version.

Cache hits are not guaranteed, since cleartext storage pools are periodically *scrubbed*. (See Chapter 10, "Periodic Maintenance of the Data Repository".) A miss simply means that the type manager must be invoked to reconstruct the version again.

Derived Object Storage Pools

Each *derived object storage pool* holds a collection of *derived object data containers*. A derived object data container holds the file system data (typically, binary data) of one DO, created during *clearmake* or *clearaudit* execution.

DO storage pools contain data containers only for the derived objects that have been shared by two or more views, through ClearCase's *wink-in* feature. Each directory element is assigned to a particular DO storage pool; the first time a DO that was created within that directory is winked-in to some view, its data container is copied to the corresponding DO storage pool. The data containers for never-shared derived objects reside in view-private storage.

Derived object pools are periodically scrubbed, as described in Chapter 10.

The *vob_server* Process

Most access to VOB storage pools goes through a ClearCase server program, the *vob_server*. This process handles data-access requests from clients, forwarded to it by the VOB's *db_server* and *vobrpc_server* processes. As with these other servers, a *vob_server* runs on host where the corresponding VOB storage directory resides. Each VOB on a host has its own dedicated *vob_server* process.

Default, Local, and Remote Storage Pools

As part of creating a new VOB, the *mkvob* command creates three subdirectories for storage pools, with a single *default* storage pool within each one:

c	directory for all cleartext pools
c/cdft	default cleartext pool
d	directory for all derived object pools
d/ddft	default derived object pool
s	directory for all source pools
s/sdft	default source pool

You can create as many additional storage pools as desired (with *mkpool*), and can adjust the assignment of elements to these pools (with *chpool*).

By default, the *mkpool* command creates new storage pools within the VOB storage directory itself. Such pools are termed *local*. For example, *mkpool -source srcpl2* creates a local pool as subdirectory *s/srcpl2* under the VOB storage directory.

You can use `mkpool -ln` to create a *remote storage pool*, leaving behind a standard UNIX-level symbolic link that points to the remote location:

```
# cleartool mkpool -source -ln /net/ccsvr04/ccase_pools/srcpl3 srcpl3
```

In this example, a storage pool directory is created at the remote location */net/ccsvr04/ccase_pools/srcpl3*. Within the VOB storage directory, a symbolic

link is created instead of a subdirectory; the text of the link is */net/ccsvr04/ccase_pools/srcpl3.*

The remote location can be on another host—even a non-ClearCase host—or in another disk partition on the local host (Figure 2-2). Either way, this capability enables a VOB to circumvent the UNIX limitation that restricts a directory tree to be wholly contained within a single disk partition. It also allows you to use high-capacity and/or high-speed file servers on which ClearCase is not installed.

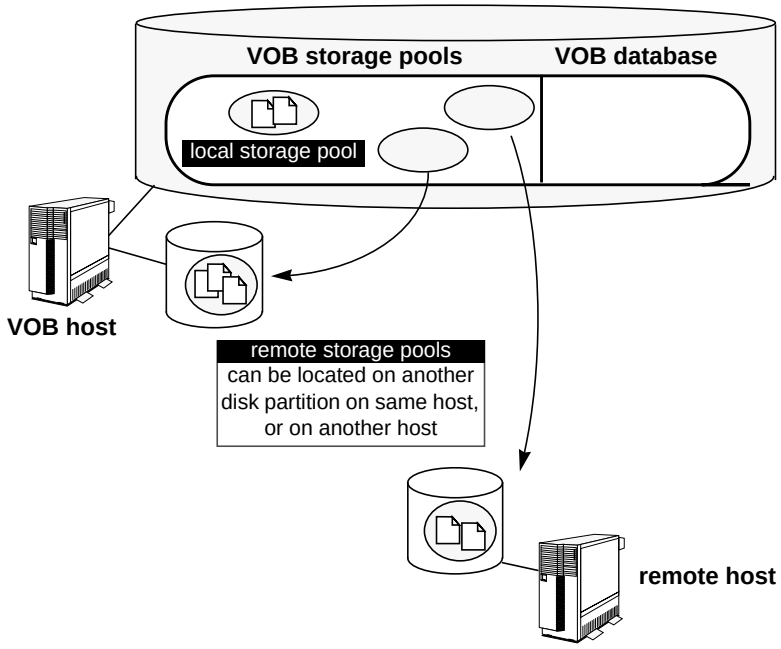


Figure 2-2 Local and Remote VOBStorage Pools

This is a powerful feature, enabling a single *logical* entity to be distributed *physically.* But there are some provisos:

- The important task of data backup is considerably harder for a distributed VOB than for a VOB wholly contained in a single disk partition. See “Backing Up a VOB with Remote Storage Pools” on page 86.

- You must be careful in devising the pathname of the remote location. This pathname must be valid on *all* client hosts that will access the VOB. In particular, you cannot use the *network region* facility to handle network idiosyncrasies, such as hosts with multiple network interfaces.

Elements' Source Pool Assignments

The *mkvob* command creates a single directory element, the VOB's *root directory*. Users access this directory at the *VOB-tag* location (the VOB's mount point). This top-level directory is assigned to the three default storage pools; and by default, all newly-created elements inherit the pool assignments of their parent directories. Thus, all elements in a VOB will use the default storage pools, unless you create new pools and reassign elements to them.

You can use the *chpool* command to change the source and/or cleartext pool associated with an element. Changing the source pool of a file moves all its data containers; for a directory element, this changes the source pool to which new elements created within it will be assigned. (See also "Creating Additional VOB Storage Pools" on page 113, and the *mkvob* manual page.)

Commands for Working with Storage Pools

The following commands are your basic tools for working with VOB storage pools. Each has its own manual page, which provides complete details on its usage.

***cleartool* subcommands:**

mkpool	Creates a new storage pool; with *-update*, adjusts an existing pool's scrubbing parameters.
lspool	Lists basic information about one or more storage pools. (The `describe -pool` command lists the same information.)
rnpool	Renames a storage pool.
rmpool	Deletes a storage pool.
chpool	Reassigns elements to a different pool.

utility commands:

scrubber	Deletes unneeded data containers from derived object and cleartext pools. (See also "Scrubbing VOB Storage Pools" on page 96.)
view_scrubber	With `-p` option, transfers data containers from view-private storage to a VOB's derived object storage pool. (See also "Scrubbing View-Private Storage" on page 99.)

Views

A ClearCase development environment can includes any number of *views*. A typical view is "private" to a single user, or perhaps to a small group of users tackling a particular task as a team.

Each view implements a *virtual workspace*, which presents its user(s) with an extended file system that superficially appears to be a standard UNIX file system hierarchy. This workspace combines:

- Selected versions of elements (actually stored in VOB storage pools)
- Files that are being modified (checked-out file elements, stored in the view's private storage area)
- Directories that are being modified (checked-out directory elements, maintained in the VOB database)
- Derived objects built by users working in this view (stored in the view's private storage area); configuration records that correspond to these derived objects
- Derived objects originally built in another view, but them *winked-in* to this view (stored in VOB storage pools)
- *View-private objects*: miscellaneous files, directories, and links that appear only in this view (stored in the view's private storage area)

Each view is a UNIX directory tree, whose top-level directory is termed the *view storage directory*. The main components of this directory tree are:

- **View database**—The *db* subdirectory contains the binary files managed by ClearCase's embedded DBMS. The database tracks the correspondence between VOB objects and view-private objects. For example, a *checkout* of a file element creates a "checked-out-version" object in the VOB database, and a corresponding data file in the view's data storage area. The view database records the relationship between these two objects.
- **Private storage area**—The *.s* subdirectory is the top-level of a directory tree in which all view-private objects are stored: checked-out versions of file elements, unshared derived objects, text-editor backup files, and so on. Each view-private file, which appears to be located in some directory within some VOB, is actually stored in a *data container* in the view's private storage area.
- **Identity directory**—The *.identity* subdirectory contains files that establish the view's owner, its principal group, and its group list.

The *view* manual page provides a detailed description of the contents of a view. (For more on the *.identity* directory, see "VOBs and Views: Owner and Groups" on page 39.)

View Database

Each view has its own database, implemented as a set of files in the *db* subdirectory of the view storage directory. On-disk overhead for the database is quite small—usually less than 1Mb.

A single ClearCase server program, the *view_server*, is invoked when the view is activated (for example, with a *startview* or *setview* command). This enables ClearCase client programs and standard UNIX programs to use the view—both to access VOB data and to access view-private data. The *view_server* process runs on the host where the view storage directory resides.

Caution: Never try to move the view database directory (*db*) to another host. But see "Moving a View (Same Architecture)" on page 121.

View's Private Storage Area

A view's *private storage area* is a subtree in the view storage directory. It provides disk storage for view-private files (including checked-out versions of file elements) and for derived objects actually built in that view. *clearmake* also caches configuration records of recently-built derived objects in this area, to speed *configuration lookup* (build avoidance).

Typically, unshared derived objects make the greatest storage demand on a view's private storage area. When a derived object is first created, both its data container file and its configuration record are stored in the view. The first time the derived object is *winked-in* to another view:

The configuration record is moved to the appropriate VOB database, or databases. If the build script creates derived objects in several VOBs, each VOB database gets a copy of the same configuration record.

- The data container is copied (not moved) to a VOB storage pool. The original data container remains in view storage, to avoid "pulling the rug out from under" user processes that are currently accessing the data container. From time to time, you (or whichever user "owns" the view) may find it worthwhile to eliminate the redundant storage containers from views with the *view_scrubber* utility. (See Chapter 10.)

A view's private storage area can be located remotely from the view storage directory, and accessed through a standard UNIX-level symbolic link. This resembles the remote VOB storage pool facility, discussed in "Default, Local, and Remote Storage Pools" on page 15, but the facility for views is less elaborate:

- A VOB can have any number of storage pools, any of which can be remote.
- A view has a single private storage area: the directory tree with *view-storage-dir/**.s*** as its root. By default, the *mkview* command creates a view storage directory with *.s* as an actual subdirectory; the *mkview -ln* command creates a view with *.s* as a symbolic link to another location.

The same restriction as for remote VOB storage pools applies: a remote private storage area must be NFS-accessible at the same pathname from all ClearCase hosts (for example, */net/cccvr02/view_storage/drp*).

If a view storage directory threatens to fill up its disk partition, you can move its *.s* directory to a larger partition. See Chapter 12, "Occasional View Maintenance" for details.

Chapter 3

Network-Wide Access to ClearCase Data

This chapter describes the mechanisms by which ClearCase data structures—VOBs and views—are made available throughout the local area network.

Storage Directories and Access Paths

Each ClearCase VOB and view has both a physical location and a logical location:

- **Physical location**—Each *VOB storage directory* and *view storage directory* is actually a directory tree, located on some ClearCase host. For day-to-day work, developers need not know the actual locations of these storage directories.
- **Logical location**—Each VOB and view also has a tag, which specifies its logical location. In their day-to-day work, developers use *VOB-tags* and *view-tags* to access the data structures.

Distributed VOBs and Views

ClearCase allows you to distribute the data storage for a given VOB or view to more than one host. You can create any number of additional *VOB storage pools* that are remote to the VOB storage directory (*mkpool* command); similarly, you can place a view's private storage area on a remote host (*mkview -ln* command).

In both cases, remote data storage is implemented at the UNIX level. As far as ClearCase servers are concerned, the data is located within the VOB or view storage directory—standard UNIX symbolic links cause the reference to "go remote".

Note: Remote data storage is outside the scope of this chapter; see Remote data storage is outside the scope of this chapter; see Chapter 2, "ClearCase Data Storage" for a discussion. In particular, the ClearCase storage registries discussed in the remainder of this chapter are not used to resolve the symbolic links that implement distributed data storage. ♦

Storage Registries

All VOB storage directories are registered in a set of files that constitute the *VOB registry*; all view storage directories are registered in files that constitute the *view registry*. These storage registries record physical locations—hostnames and pathnames on those hosts; they also record the logical *access paths* used by clients and servers to access VOB and view data.

A storage registry has two parts, implemented in separate files: an *object registry* and a *tag registry*. The following sections provide an overview of these components; for details, see the *registry_ccase* manual page.

Object Registries

The *VOB object* and *view object* registries record the location of each VOB and view using a host-local pathname. That is, the pathname to the data structure is one that is valid on the host where the storage directory resides. These pathnames are used by the ClearCase server processes (*view_server*, *vob_server*, and so on), which run locally.

An entry is placed in the appropriate object registry when a VOB or view is first created (*mkvob*, *mkview*). The entry is updated automatically whenever a reformatting is performed (*reformatvob*, *reformatview*); you can also update or remove the entry manually (*register*, *unregister*).

Object registry entries are used mostly by ClearCase server processes.

Tag Registries

For most purposes (including virtually all day-to-day development activities), VOBs and views are not referenced by their physical storage locations. Instead, they are referenced by their VOB-tags and view-tags:

- A VOB's *VOB-tag* is its mount point as a file system of type MVFS. Developers access all ClearCase data (MVFS files and directories) at pathnames below VOB mount points.
- A view's *view-tag* appears as a subdirectory entry in a host's *viewroot* directory, */view*. For example, a view with tag *oldwork* appears in the host's file system as directory */view/oldwork*. To access ClearCase data, developers must use a view—either implicitly (by setting the view) or explicitly (by using a view-extended pathname).

Thus, any reference to a ClearCase file system object involves both a VOB-tag and a view-tag. These logical locations are resolved to physical storage locations through lookups in the network-wide *VOB-tag* and *view-tag* registries. Each tag registry entry includes a *global pathname* to the storage area—a pathname that is valid on all ClearCase client hosts. Figure 3-1 illustrates how tag registries and object registries are used to access the network's set of data storage areas.

Note: In some networks, it is not possible to devise global pathnames to all ClearCase storage areas. The ClearCase *network region* facility handles such situations—see "Network Regions" on page 28. For simplicity, Figure 3-1 illustrates a network that has a single network region. ♦

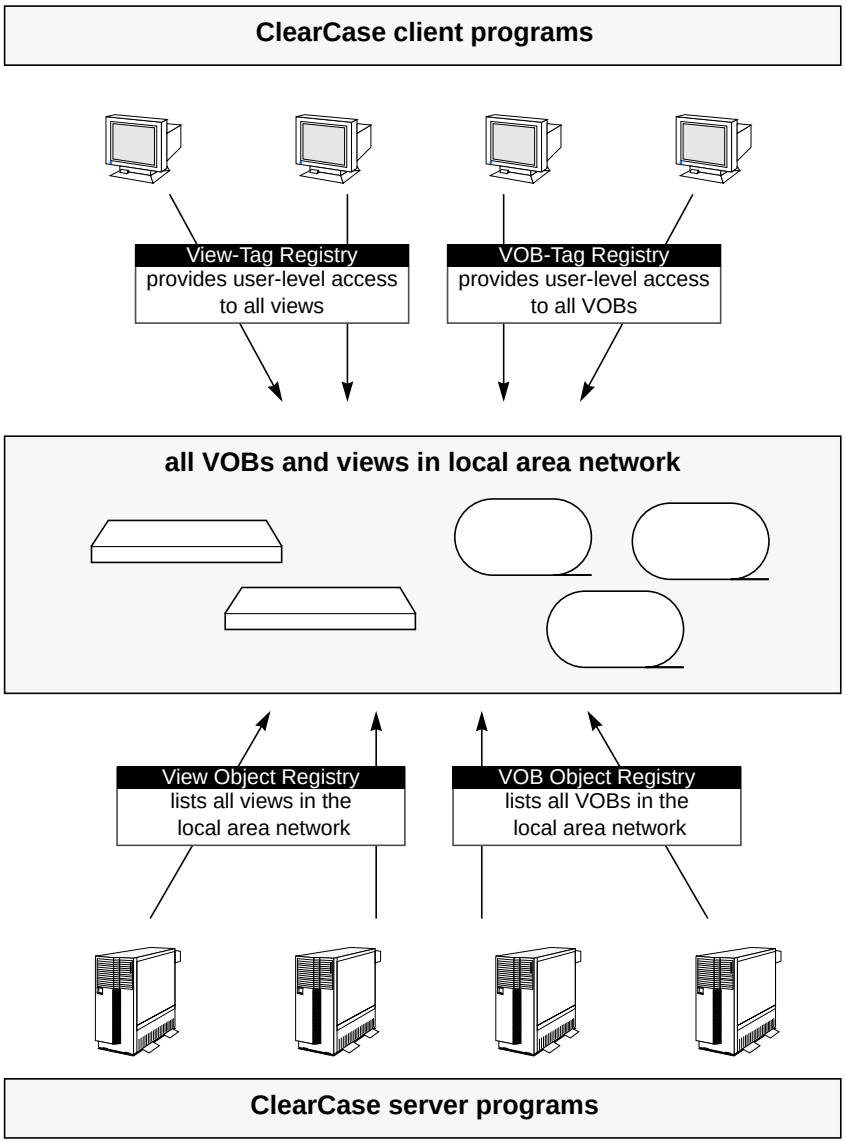


Figure 3-1 ClearCase Object and Tag Registries (Single Network Region)

Network-Wide Accessibility of VOBs and Views

ClearCase's network-wide storage registries make all VOBs and views visible to all users. You can use the *lsvob* and *lsview* commands to list them all. But typically, VOBs and views have different usage patterns:

- Most users require access to most (or all) VOBs.
- Most users need to access only a small number of views.

Accordingly, there are different schemes for *activating* VOBs and views on each client host. A set of *public* VOBs is activated automatically by the ClearCase startup script on a client host. By contrast, no views are activated automatically at ClearCase startup; the user(s) on a client host must activate their view(s) with explicit commands.

Public and Private VOBs

To provide control over which VOBs are activated automatically, each VOB is designated as *public* or *private* when it is created. More precisely, each VOB-tag is either public or private. Only public VOBs are activated automatically; a private VOB becomes active only when its owner enters an explicit mount command.

Public VOBs can be activated and deactivated (mounted and unmounted) by any ClearCase user. The actual mounting is performed by a short-lived server process, *mntrpc_server*, which runs as the *root* user. A password facility controls creation of these mountable-by-anyone data structures: when a VOB-tag is created (during execution of a *mkvob* or *mktag -vob* command), you must enter a password to match the one stored in the *VOB-tag password* file: */usr/adm/atria/rgy/vob_tag.sec* on the registry server host.

Note: Be careful when making public VOBs. Each ClearCase client host will attempt to mount all public VOBs whenever the operating system is started (and whenever ClearCase processing is restarted with an explicit command). ♦

Network Regions

Ideally, your network's VOB and view storage directories should be accessible at the same pathnames throughout the network. Automatic file-system mount utilities, such as *automount*(1M), are intended to achieve the ideal of uniform, global naming. Figure 3-2 shows a simple network in which global naming has been achieved.

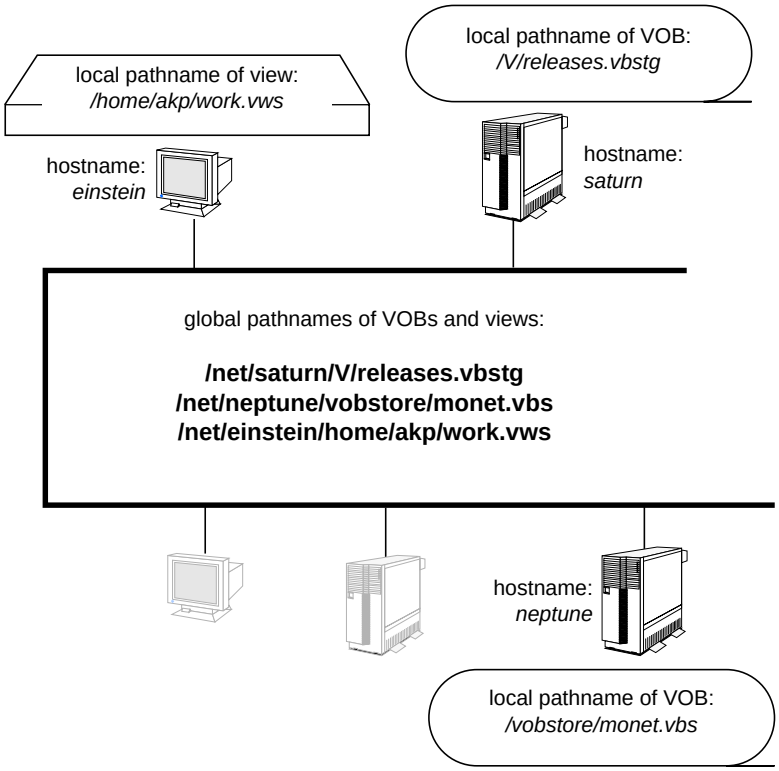


Figure 3-2 Network with Global Naming

Uniform, global naming may not be achievable, however. The most common reasons are:

- **Multiple network interfaces**—A VOB host or view host may have two or more interfaces to the network, each corresponding to a different UNIX-level hostname. For example, a host might be known to some

hosts (and their automounter programs) as *neptune*, and to other hosts as *neptune-gw*. (The "gw" suffix is commonly used, standing for "gateway".) In this case, the same VOB might have two "global" storage pathnames:

```
/net/neptune/public/project.vbs
/net/neptune-gw/public/project.vbs
```

- **Multiple aliases**—The standard UNIX facilities for assigning names to hosts—file */etc/hosts* or NIS map *hosts*—allow each host to have any number of alternate names, or *aliases*. This is a possible hosts entry:

  ```
  195.34.208.17 betelgeuse bg      ("gratuitous" alias)
  ```

  If shared storage resides on this host, ClearCase clients might be able to access the storage using either a "/net/betelgeuse/..." pathname or a "/net/bg/..." pathname.

- **Multiple architectures**—A heterogeneous network may include hosts that support very different file systems. For example, a VOB that is accessed as */net/neptune/vobstore/incl.vbs* on a UNIX host may be accessed as *X:\vobstore\incl.vbs* on a Windows/NT host.

ClearCase servers require consistent pathnames to shared storage areas. If you cannot achieve *global* consistency, then you must partition your network into a set of *network regions*, each of which is a consistent naming domain:

- Each ClearCase host must belong to a single network region.
- All hosts in a given network region must be able to access ClearCase physical data storage (that is, all VOB storage directories and the storage directories of shared views) using the same full pathnames.
- Developers access VOBs and views through their VOB-tags (mount points) and view-tags. All hosts in a given network region use the same tags.

For example, a VOB and a view might be accessed in different network regions as follows:

Region: *core_dvt*
VOB storage: */net/neptune/public/vega_project.vbs*
VOB-tag: */vobs/vega*
View storage: */net/saturn/shared_views/int_43.vws*
View-tag: *int_43*

Region: *lib_dvt*
VOB storage: */net/neptune-gw/public/project.vbs*
VOB-tag: */vobs/vega*
View storage: */net/saturn-gw/shared_views/int_43.vws*
View-tag: *int_43*

Registries in a Multiple-Region Network

Conceptually, each network region has its own *view-tag registry* and *VOB-tag registry.* Each VOB can have at most one tag in a region; views can have multiple tags in a region. In a typical network with *N* regions, each VOB or view storage directory has *N* tag entries.

Note: A VOB or view need not have a tag in every region. However, a VOB or view is inaccessible for development work on hosts in any region for which it is "tagless". This suggests that you might use network regions as "access domains" instead of "naming domains". ♦

If possible, keep the tag itself constant over all the regions. For example:

Region	VOB-tag	Pathname to Storage Area in Region
uno	*/vobs/project*	*/net/neptune/public/vega_project.vbs*
dos	*/vobs/project*	*/net/neptune-gw/public/vega_project.vbs*
tres	*/vobs/project*	*/netstorage/vega_project.vbs*

This set of tags provides a single developer-visible name for the VOB (*/vobs/project*), even though network file system idiosyncrasies require several different names for the VOB's physical storage location.

Tag Registry Implementation

All view-tag registries are actually implemented in a single file, *view_tags*, on the registry server host. Each view-tag entry has a region field, which places the entry in a particular region. Similarly, a single *vob_tags* file implements all the logically distinct VOB-tag registries.

Figure 3-3 illustrates a simple two-region network, each with its own logical set of tag registries. All hosts in a network region use the same VOB-tags and view-tags, and access ClearCase data storage areas using the same pathnames, provided by registry lookups.

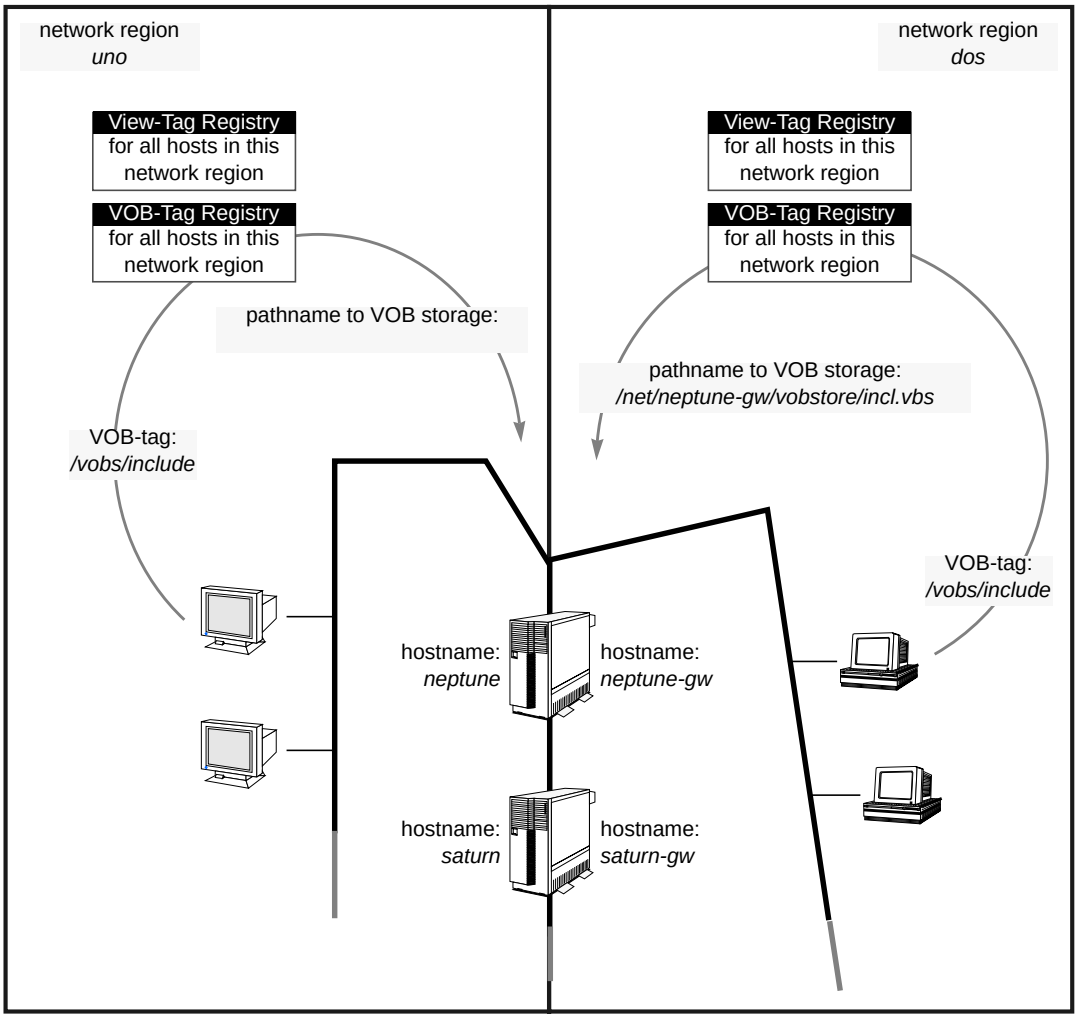


Figure 3-3 Network Regions and Their Tag Registries

Establishing Network Regions

Just after you load a ClearCase release from its distribution medium, you run a *site_prep* program. This program prompts you to specify the name of a network region. This name becomes the default region, which can be accepted or overridden during ClearCase installation on individual hosts. A host's network region assignment is recorded in file */usr/adm/atria/rgy/rgy_region.conf* on that host.

There is no formal mechanism for "defining" additional network regions. Nor is there any centralized list of region names or assignments of hosts to regions. For procedures relating to network regions, see Chapter 18, "Adjusting ClearCase Registry Information".

Recording Multiple Network Interfaces

Note: This section applies to *all* ClearCase hosts, not just to hosts where VOB and view storage directories reside. ♦

If a host has two or more network interfaces (two or more separate lines in the */etc/hosts* file or the *hosts* NIS map), it must have a file called */usr/adm/atria/config/alternate_hostnames*, which records its multiple entries. For example, suppose that the */etc/hosts* file includes these entries:

```
 .
 .
159.0.10.16 widget sun-005 wid
 .
159.0.16.103 widget-gte sun-105
 .
```

In this case, the *alternate_hostnames* file should contain:

```
widget
widget-gte
```

Note that only the first hostname in each *hosts* entry need be included in the file. In general, the file must list each alternative hostname on a separate line. There is no commenting facility—all lines are significant. If a host does *not* have multiple network interfaces, this file should not exist at all on that host.

ClearCase Data and Non-ClearCase Hosts

In large development shops, some groups might adopt ClearCase before others. There is no problem with such "incremental adoption"—a host on which ClearCase has not yet been installed can still mount VOBs and access their data.

- A ClearCase host must use file */etc/exports.mvfs* to explicitly export a view-extended pathname to the VOB mount point (for example, */view/exportvu/vobs/vegaproj*).
- One or more non-ClearCase hosts mount the VOB through a view-extended pathname. For example, a host might have an entry in its file system table that begins:

  ```
  mars:/view/exportvu/vobs/vegaproj /usr/vega nfs ...
  ```

Usage Restrictions

Users on the non-ClearCase host can only read data from such VOBs—they cannot modify the VOB in any way. Moreover, they are restricted to using the element versions selected by the specified view. They cannot use version-extended or view-extended pathnames to access other versions of the VOB's elements.

There are techniques for relaxing these restrictions in practice. A user who also has an account on the ClearCase host can reconfigure the "mounted" view, by performing an *rlogin*(1) there and modifying the view's config spec. The same VOB can be mounted at several locations on a non-ClearCase host, each mount using a different view.

Building on a Non-ClearCase Host

Although users cannot modify VOBs that are mounted through a view, they can write to view-private storage. This enables editing and building—with a native *make* program or with scripts, not with *clearmake*. Files created by builds in the VOB's directories do not automatically become derived objects; they will be view-private files, unless developers take steps to convert them to derived objects. (For more on this topic, the *CASEVision/ClearCase User's Guide*.)

Since *clearmake* does not run on the non-ClearCase host, configuration lookup and derived object sharing are not available to the *make* utility or script that performs the native build.

Using *automount* with ClearCase

This section discusses use of the standard UNIX *automount*(1M) program with ClearCase. Implementation of the facility vary from architecture to architecture; be sure to consult the documentation supplied by your hardware vendor.

Use of *–hosts* Map Required

You can use any *automount* maps, including both "direct" and "indirect" maps, to access remote disk storage where VOB storage areas reside. For proper ClearCase operation, every ClearCase host must *also* use the special "–hosts" map to provide paths to remote VOB and view storage. ClearCase looks for symbolic links to the mount points created through the "–hosts" map in any of these directories:

/net (the *automount* default)

/hosts

/nfs

If your site uses another directory for this purpose (for example, */remote*), create a UNIX symbolic link to provide access to your directory through one of the expected pathnames. For example:

```
# ln -s /remote /net
```

Specifying a Non-Standard Mount Directory

By default, *automount* mounts directories under */tmp_mnt*. If a ClearCase host uses another location for a host's automatic mounts (for example, you use `automount -M`), you must specify it in file */usr/adm/atria/config/automount_prefix*. For example, if your automatic mounts take place within directory */autom*, place this line in the *automount_prefix* file:

```
/autom
```

Chapter 4

User-Level Access to ClearCase Data

"Smooth running" with ClearCase requires careful coordination of:

- the *user base*: user-IDs and group-IDs
- ClearCase data repositories (VOBs)
- private and shared ClearCase workspaces (views)

This chapter discusses the architecture of the user base, along with the UNIX-compatible access-permissions model for VOBs, views, and the file system objects they contain. We also discuss ClearCase-specific mechanisms, which control access to VOB databases.

(Procedural details on implementing your desired data-access strategy are deferred to Chapter 6, "Setting Up ClearCase VOBs".)

Users: Usernames and Groups

ClearCase relies on standard UNIX facilities for identifying users and assigning them to groups:

- Each user has a *username* ("login" name) and a *principal group*, which are established by an entry in the password database.
- The user can also belong to any number of additional groups; the *group list* is specified by entries in the group database.

Each host has a local *passwd*(4) and *group*(4) file. Many organizations supplement these local files with network-wide NIS maps, named *passwd* and *group*. Figure 4-1 shows how a user named *derek* gets his principal group and an additional group-list assignment.

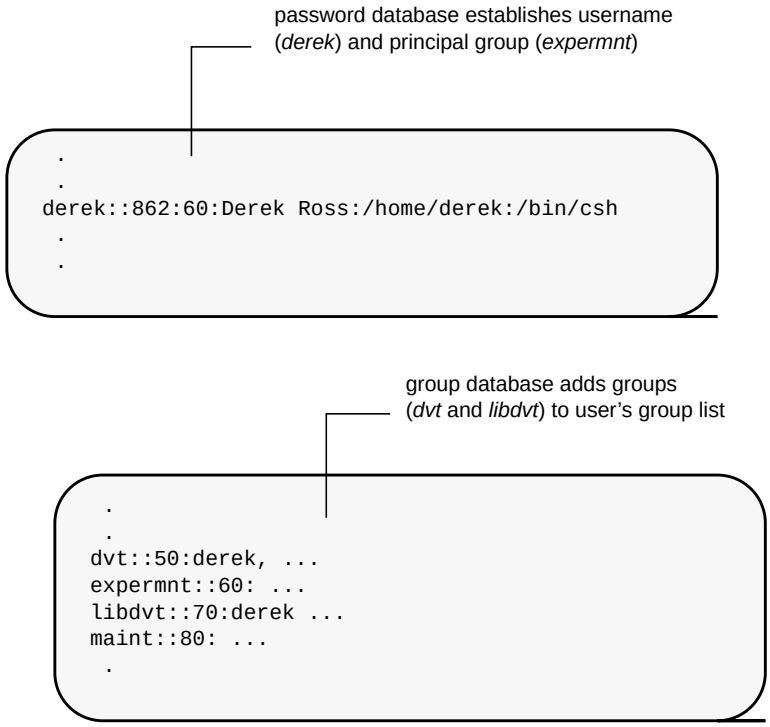


Figure 4-1 Establishing a User's Group Assignments

Note: On HP-UX hosts, the file */etc/logingroup* establishes a user's initial group assignments. See *group*(4) for details. ♦

Since ClearCase is a distributed application, it is essential that user/group identities be consistent across the network. For example, ClearCase will not work properly if a developer has user-ID 453 on one host, and user-ID 309 on another host.

Network-Wide ClearCase Administrator

We recommend that you create a "ClearCase administrator" identity in your network's user base, and that all shared ClearCase data structures—VOBs and shared views—be owned by that user. This is not a requirement, but it will facilitate ClearCase administration.

In the remainder of this manual, we assume that a ClearCase administrator, *vobadm*, has been created. If most developers belong to a single group, make *vobadm* a member of that group, too.

Note: You may be tempted to let *root* perform all ClearCase administrative tasks. But many organizations prefer to have a separate identity for application-specific administration. Moreover, *root* is often not the "same user" throughout the network—one host's *root* is another host's *nobody.* ♦

VOBs and Views: Owner and Groups

For purposes of determining access permissions, each ClearCase VOB and view has an *owner*, a *principal group*, and an optional *group list*. (Note the similarity to the user-identification structures introduced in the preceding section.) The multiple-group facility for these data structures supports development environments where (1) users don't all belong to the same group, and (2) non-group members are prohibited from accessing data:

- *Rule:* For a user to have permission to read a VOB's data, one of the user's groups must be the VOB's principal group.
- *Rule:* To read a VOB's data through a particular view, one of the *view_server*'s groups must be the VOB's principal group.
- *Rule:* For a user to have permission to modify a VOB's data, the user's principal group must be (any) one of the VOB's groups.

Figure 4-2 illustrates these access paths; the mechanics are discussed in "Read Access by Processes" on page 51.

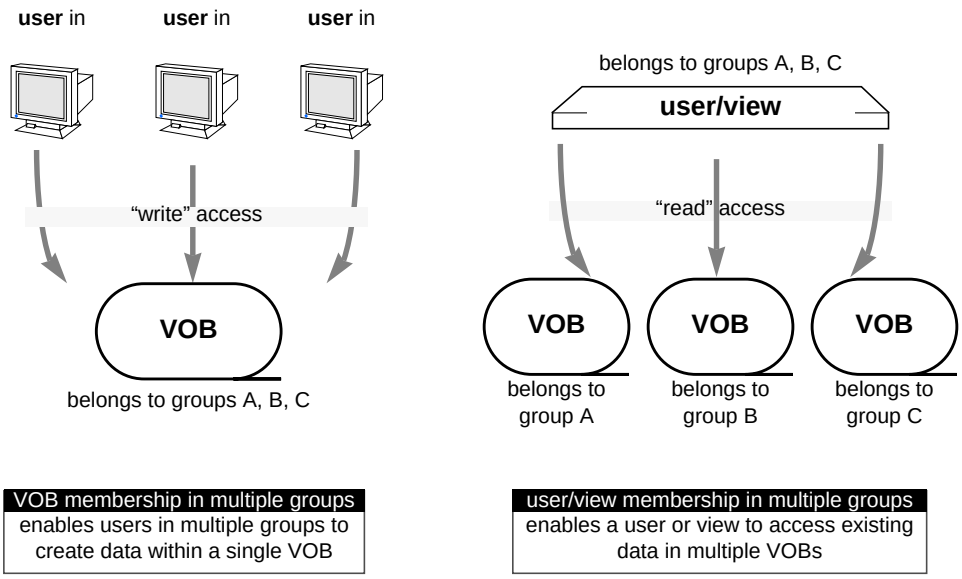


Figure 4-2 Multiple-Group Support for VOBs and Views

The .identity Directory

At the operating system level, each file system object has a single group, not a list. Accordingly, a ClearCase-specific mechanism is required to implement the group list of a VOB or view. Each VOB storage directory and view storage directory has a *.identity* subdirectory. Entries in this directory establish the *group list*, and also the *owner* and *principal group* (Figure 4-3).

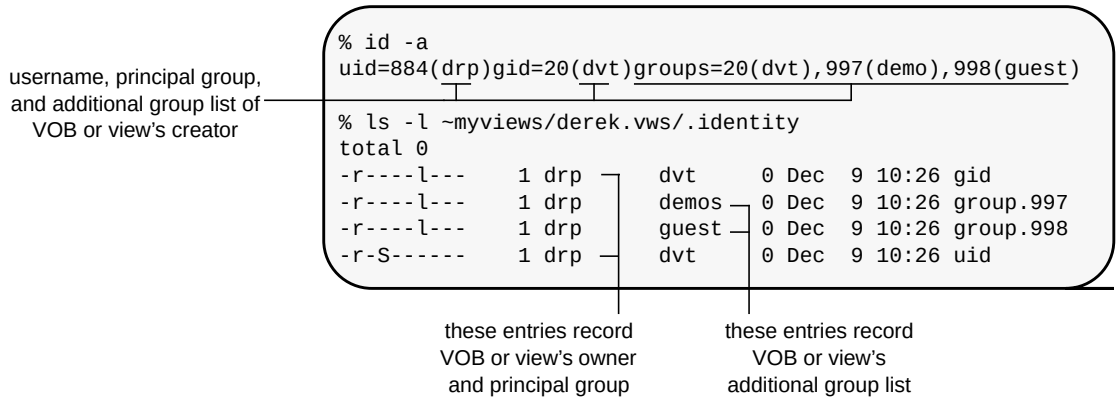


Figure 4-3 The .identity Directory of a VOB or View

Use the `describe -vob` command to list a VOB's identity information. (There is no "-view" option to the *describe* command; use a standard `ls -l` command on the *.identity* subdirectory of the view storage directory.)

To modify a VOB's identity information, use the *protectvob* command. You must be the *root* user to use this command:

```
% su
Password: <enter root password>
# cleartool protectvob -add_group demos,guest
/vobstore/proj.vbs

 <messages and verification prompts>

VOB ownership:
  owner drp
  group dvt
Additional groups:
  group demos
  group guest
```

Caution: Do not try to change this information directly, using standard UNIX commands.

Note: There is no comparable command for modifying a view's identity information. ♦

Effect of Creator's 'umask' Setting

A user's current *umask*(1) setting affects the way in which a *mkvob* or *mkview* command creates a new data structure. In general, a more restrictive umask produces a VOB or view that has more limited accessibility:

- For a new VOB, the user's umask determines the access permissions on the VOB's top-level directory element—its *root directory.* (The umask is subtracted from mode 777, in the standard manner.) This directory is the "gateway" to the entire VOB, and thus provides an important access-control point. To access any file system object that appears in that VOB (including view-private objects created in VOB directories), a user process must have permission to traverse the VOB's directory element hierarchy, starting with the VOB root directory.
- For a new view, the user's umask determines the access permissions on the view storage directory itself, and on all directories within the view's private storage area. (Again, the umask is subtracted from mode 777, in the standard manner.) To access a view-private object, a user process must have permission to traverse the directory structure in the view's private storage area.

Note that access to view-private objects is subject to two kinds of permissions checking: on the VOB's logical hierarchy of directory elements, and on the view's physical hierarchy of directories within its private storage area.

Example: Two VOBs and Two User Groups

Suppose that you wish to place two source trees *libpub* and *monet*, into separate VOBs, to be accessed by two groups, *dvt* and *mon*, as shown in Figure 4-4.

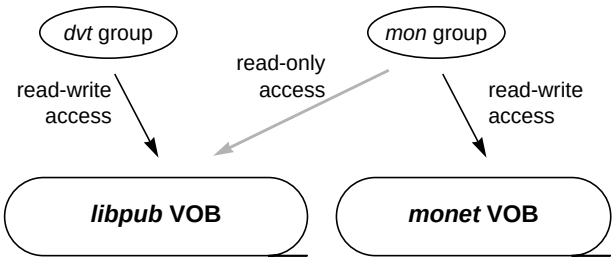


Figure 4-4 Planning Access to Development Sources

As suggested in "Network-Wide ClearCase Administrator" on page 39, suppose that you are the VOB administrator, belonging to both groups. (It really doesn't matter which of these is your principal group, and which is established in the *group* database.)

```
vobadm:@!bork%$:555:30:VOB administrator:/:/bin/csh
```
(/etc/passwd or NIS passwd map entry: group 30 is dvt)
```
mon::35:vobadm
```
(/etc/group or NIS group map entry: group 35 is mon)

Creating the VOBs

The following procedure creates the VOBs and adjusts their permissions. (For more on creating VOBs, see Chapter 6, "Setting Up ClearCase VOBs".)

1. **Prepare a physical storage location**—(This is the only step that requires *root* privileges. If you do not have *root* access, you can enlist the aid of someone who does, for this step only.) You will create VOB storage directories within a globally-accessible directory tree, at */vobstore* on host *sol*.

   ```
   % su
   Password: <enter root password>
   # mkdir /vobstore
   ```

   With many UNIX variants, the disk partition mounted as the root file system (/) is quite small, and will not accommodate ClearCase VOBs. In such cases, create the actual directory in a large partition, and link it to */vobstore*. For example:

```
# mkdir /usr/vobstore
# ln -s /usr/vobstore /vobstore
# exit
```

2. **AssumSte the appropriate user identity**—Make sure that you are logged in as user *vobadm*, in group *dvt*.

```
% newgrp dvt
% id
uid=555(vobadm) gid=30(dvt)
```

3. **Create the first VOB storage directory**—Be sure to use the location you prepared in Step #1. This command creates the *libpub* VOB.

```
% cd /vobstore
% cleartool mkvob -tag /vobs/libpub -public libpub.vbs
Vob tag registry password: <enter password>
Comments for "libpub.vbs":
Sources for libpub.a library
.
Created versioned object base "libpub.vbs".
VOB ownership:
  owner vobadm
  group dvt
```

4. **Switch your principal group**—Remain user *vobadm*, but group *mon* your principal group.

```
% newgrp mon
% id
uid=555(vobadm) gid=35(mon)
```

5. **Create the second VOB storage directory**—This command creates the *monet* VOB.

```
% cleartool mkvob -tag /vobs/monet -public monet.vbs
Vob tag registry password: <enter password>
Comments for "monet.vbs":
Sources for monet project
.
Created versioned object base "monet.vbs".
VOB ownership:
  owner vobadm
  group mon
```

Controlling Access to the VOBs

You (*vobadm*) are the *owner* of both new VOBs, by virtue of having created them. By changing groups with the *newgrp* command, you created VOBs that belong to different groups. In general, all members of the *dvt* group will have complete access to the *libpub* VOB; likewise, all members of the *mon* group will have complete access to the *monet* VOB.

To complete implementation of the VOB-access scheme depicted in Figure 4-4, the only necessary adjustment is dictated by the first rule in "VOBs and Views: Owner and Groups" on page 39, all members of the *mon* group must have *dvt* as an additional group in order to enable read access to the *libpub* VOB:

```
dvt::30:allison,david,ralph
```
(all members of the mon group ....
... get group 30 (dvt—the libpub VOB's principal group) as an additional group assignment in the /etc/group file or NIS group map)

Access to Individual File System Objects

On a day-to-day basis, developers deal with individual file system objects, rather than entire VOBs and views. In general, the standard UNIX access-permissions model applies to ClearCase-controlled file system data. Files, directories, and links stored in VOBs and views all have *stat*(2) records, which contain the familiar *user-group-other* and *read-write-execute* information.

Access-Control Settings

To standard UNIX operations, a VOB appears to be a directory tree, containing directories, files, and links. Each of these file system objects has a single *owner*, a single *group*, and an *access mode.* That is, its *access-control settings* seems completely "ordinary" to the `ls -l` command:

```
% ls -l
total 50
-r--r--r--    1 drp      dvt          224 Feb 19 15:34 Makefile
-rwxrwxr-x    1 drp      dvt        14956 Feb 19 15:34 hello
-r--r--r--    1 drp      dvt           79 Feb 19 15:34 hello.c
```

```
-r--r--r--    1 drp      dvt          168 Feb 19 15:34 hello.h
-rw-rw-r--    1 drp      dvt         1504 Feb 19 15:34 hello.o
-r--r--r--    1 drp      dvt          232 Feb 19 15:34 msg.c
-rw-rw-r--    1 drp      dvt         2168 Feb 19 15:34 msg.o
-r--r--r--    1 drp      dvt          511 Feb 19 15:33 util.c
-rw-rw-r--    1 drp      dvt         3228 Feb 19 15:34 util.o
```

(Depending on the UNIX variant, the `-g` option to the `ls` command either turns off display of group memberships, or turns it on.)

What the developer sees is a combination of view-resident objects and VOB-resident objects (the view's *virtual workspace*). The distinction is important for this discussion, because different access-control mechanisms apply:

- Each element and all of its versions have the same access-control settings. The *protect* command controls these settings, with `-chown`, `-chgrp`, and `-chmod` options. The `-chmod` option controls the "read" and "execute" (`r` and `x`) fields only, not the "write" (`w`) field—all versions in VOB storage are *always* read-only to standard UNIX commands.

  You might complain that *checkout* creates a version that is writable, not read-only. But the checked-out version that a user edits is not a VOB object—it is a view-private file. All the versions in VOB storage (that is, in the VOB's source storage pools) remain read-only after a *checkout*.

- Each view-private object has a standard UNIX access-control settings. The standard *chown*(1), *chgrp*(1), and *chmod*(1) commands control these settings.

Notes on VOB/View Interactions

The mechanisms for changing access modes work intuitively. There are some special considerations, involving situations where both a VOB and view are involved:

- The `cleartool protect -chmod` command is independent of the standard *chmod*(1) command. If a file is checked-out, changing the access mode of the *element* does not affect the mode of the view-private file which is the checked-out version.

- The `cleartool protect -chmod` command cannot be used on an unshared derived object. Using this command on a shared derived object does not affect the current view, even if that derived object appears in the view. The standard *chmod* command can be used on any derived object that appears in a view—shared or unshared.
- To create a new view-private file, directory, or link, a user must have "write" permission in the parent directory, which is a VOB object (a directory element).

Initialization of Access-Control Settings for VOB Objects

When a new element or VOB symbolic link is created (with *mkelem*, *mkdir*, or *ln -s*), its access-control settings are initialized in the standard UNIX manner:

- The creator of the element or link becomes its *owner*.
- The creator's principal group becomes the group of the element or link. (In some UNIX variants, a new object gets the group of its parent directory; but ClearCase *always* uses the creator's group.)
- The creator's current *umask* determines the access mode of the element or link.

Initialization of Access-Control Settings for View Objects

When a view-private object is created, its access-control settings are initialized in the same way, from the creator.

Access-Control Settings for Physical Data Storage

All of the access-control information described in the preceding section is maintained in VOB and view databases. But ultimately, version-controlled data is stored in *storage pool* directories and *data container* files. (See Chapter 2, "ClearCase Data Storage".) These are ordinary UNIX file system objects and, as such, are subject to standard UNIX access controls.

Note: Users don't reference these objects directly, and so should not be concerned with their access permissions. ♦

ClearCase manages the permissions on storage pools and data containers automatically, preventing accidental (or mischievous) deletion of VOB data:

- All storage pools and data containers are owned by the *VOB owner* and belong to the VOB's *principal group*. (This is also the principal group of the VOB owner, unless it has been "given away" with a `cleartool protectvob -chgrp` command.)
- All storage pools have access mode 755, allowing them to be read and searched by anyone.
- A version's data container has the same access mode as the version itself (which, in turn, has the same access mode as the element). Both the version and its data container are maintained in a read-only state.

All changes to a VOB's physical data storage is performed by its dedicated *vob_server* process, which runs as the VOB owner. As appropriate, this process temporarily makes certain data structures writable, performs the change, then restores the data structures to their read-only state.

How Processes Access ClearCase Data

On UNIX systems, all data access is performed by *processes*, which run with certain identities:

- A ClearCase client process (*cleartool*, *xclearcase*, *clearmake*, and so on) runs with the user's UID and with all of the user's groups—principal group and entire group list.
- All access to ClearCase data must go through a view. That is, data is accessed by the associated *view_server* process, which runs with the UID and all the groups of the view's owner.

Most often, users work in their own "private" views, which they have created themselves. This simplifies the access-control issues, since the user's ClearCase client processes and the *view_server* process run under the same identity. The effective question is:

Does the user have permission to access the VOB data?

When users work in shared views, or attempt to “peek into” other users’ views, the access-control issues become more involved. Some or all of the following access paths are traversed when a user attempts to use ClearCase data:

- The user’s process accesses data in view storage.
- The user’s process accesses data in VOB storage.
- The view’s *view_server* process accesses its view-private storage.
- The view’s *view_server* process accesses VOB storage.

Figure 4-5 illustrates these access paths, all of which are subject to permissions checking. Note that the view is “in the middle”—it is both a data repository, accessed by a user process, and a process that accesses VOB storage.

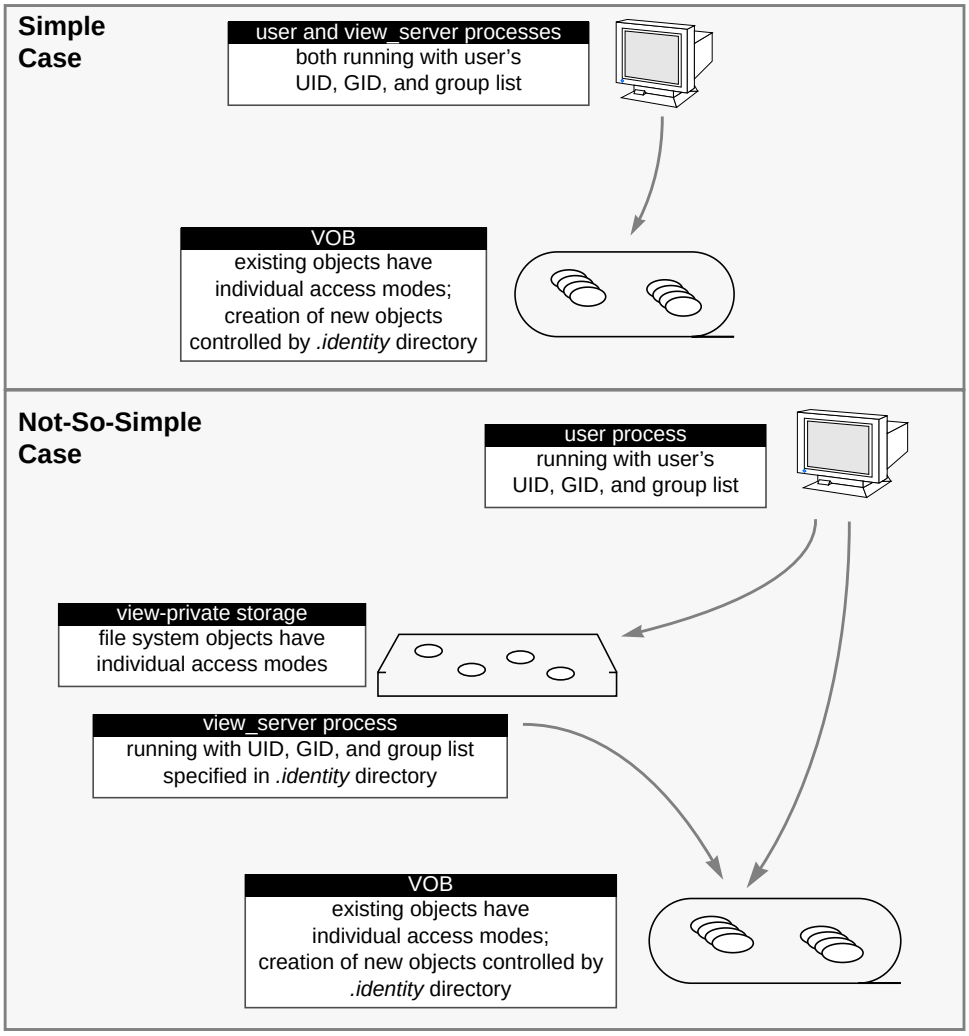


Figure 4-5 Data Access Paths

Usually, the owner of a file or directory has no trouble accessing it. The following sections apply in situations where access to data is to be granted at the group level—that is, where (1) users don't all belong to the same group, and (2) non-group members are prohibited from accessing data.

Read Access by Processes

A process gains "read by group member" access to VOB or view data if *any* of the process's groups matches the group of the data (Figure 4-6). Note that the process (potentially) has multiple groups; the data file or directory necessarily has a single group. In addition, there is a VOB-level restriction: one of the process's groups must be the VOB's principal group.

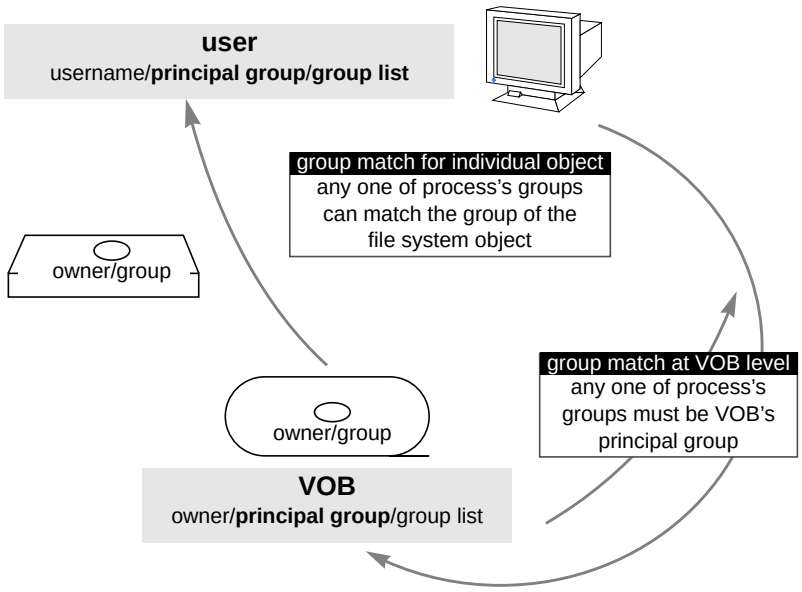


Figure 4-6 Read' Access through Group Membership

Write Access by Processes

ClearCase's mechanism for determining who can modify VOB data is finer-grained than the standard UNIX mechanism. (See "ClearCase-Level Access Permissions" on page 52.) But there is also an overall VOB-level restriction: in order to create a new element or VOB symbolic link, a user's principal group must be (any) one of the VOB's groups. This follows from the fact that *all* file system objects in the VOB must belong to one of the VOB's groups.

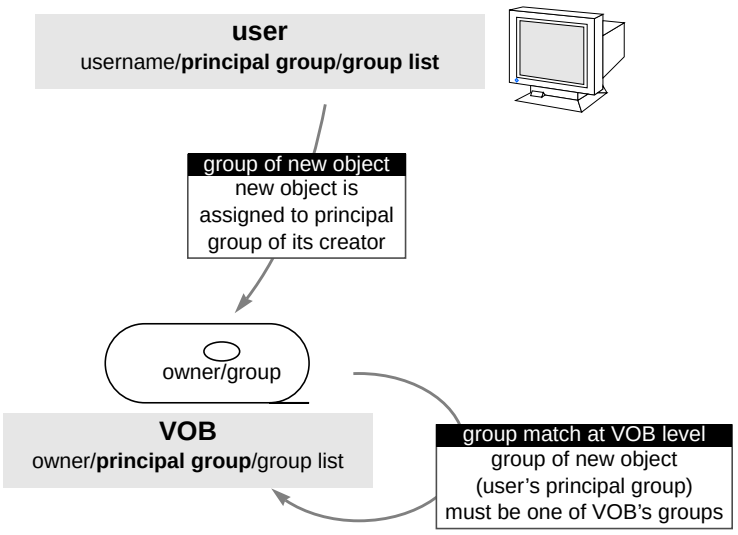


Figure 4-7 'Write' Access through Group Membership

There are no comparable group-level restrictions on the creation of new view-private objects.

ClearCase-Level Access Permissions

The standard UNIX model for handling the "write" (*w*) permission is insufficient for ClearCase's needs. ClearCase relies on a more elaborate hierarchy to determine what in a VOB can be modified, and by whom:

- the *root* user (superuser)
- the VOB owner (that is, the user who owns the VOB storage area. The user who creates a VOB becomes its owner. A subsequent *chown_vob* command changes the owner.)
- the owner of the corresponding element (for modifications to branches and versions)
- the creator of a type object (for modifications to these objects)
- the creator of a particular storage pool

- the user associated with a particular event
- the creator of a particular version or derived object
- members of an element's or derived object's group

The *root* user and the VOB's owner can perform all operations that modify a VOB. Lower levels of the hierarchy have fewer permissions, and not all levels are relevant to every command.

Whenever you enter a *cleartool* command that modifies one or more VOB objects, this permissions hierarchy is applied automatically to each object, in a command-specific manner. For example:

- Permission to remove an element with the *rmelem* command is granted to root, to the VOB owner, and to that element's owner. No one else can remove that element.
- Permission to *checkout* an element is granted to all of the same users as in the preceding example, and also to users in the element's group. The command fails if anyone else tries to execute it.

See the *ct_permissions* manual page for details on how individual commands fit into the permissions hierarchy.

Locks on VOB Objects

The ClearCase permissions scheme is intended for use as a long-lived access-control mechanism. ClearCase also provides for temporary access control, through explicit *locking* of individual VOB objects. You can use the *lock* command to restrict or completely prohibit changes at various levels. At the lowest level, you can lock an individual element, or even an individual branch of an element. At the highest level, you can lock an entire VOB, preventing all modifications to it.

When an object is locked, it cannot be modified by anyone, even by *root,* the VOB owner, or the user who created the lock. (But these users have permission to unlock the object.) The lock command accepts an optional exception list, specifying users for whom the object will not be locked.

Locking Type Objects

As an administrator, you will often find it useful to lock *type* objects; this prevents changes to the instances of those types. For example:

- You might lock the branch type *main* to all but a select group of users. This would allow the select group to perform integration or release-related cleanup work on the *main* branches of all elements. All other users can continue to work, but must do so on subbranches, not on the *main* branch.
- Locking the label type *RLS2.3* prevents anyone from creating or moving that label.

Chapter 5

ClearCase User Licensing Scheme

On Silicon Graphics platforms, ClearCase licensing is handled through NetLS. Refer to Chapter 7 of the *CASEVision/ClearCase Release Notes* to learn more about ClearCase licensing. Read the *NetWork License System Administration Guide* to learn more about NetLS.

Floating License Architecture

ClearCase implements an "active user" floating license scheme. To use ClearCase, a user must obtain a license, which grants the privilege to use ClearCase commands and data on any number of hosts in the local area network. When a user runs a ClearCase program, it attempts to obtain a license. If it gets one, the user can keep it for an extended period: entering any ClearCase command automatically renews it; but if the user doesn't enter any ClearCase command for a substantial period—by default 60 minutes—another user can take the license.

Chapter 6

Setting Up ClearCase VOBs

This chapter presents a procedure for setting up your network's ClearCase data repository—a set of globally-accessible VOBs. The outline of this chapter makes a handy checklist:

- Select a host for a new VOB
- Modify operating system resources, if necessary
- Create the VOB (and, if necessary, adjust its registry and identity information, in order to ensure global accessibility and implement access controls)
- Synchronize the VOB with other existing VOBs
- Populate the VOB with new or existing development data

The next chapter discusses the (quite similar) procedure for setting up ClearCase views.

Selecting a VOB Host

A host on which one or more VOB storage directories reside is termed a *VOB host*. A typical network distributes its VOBs among several VOB hosts. Any host with a ClearCase-supported hardware/software architecture *can* be a VOB host; selecting an *appropriate* host (and making necessary adjustments to it) is crucial to obtaining satisfactory ClearCase performance.

The most important criteria for selecting a VOB host are:

- **Main memory (RAM)**—The minimum recommended main memory size is **64Mb**. This is the most important factor in VOB performance; increasing the size of a VOB host's main memory is the easiest (and most cost-efficient) way to make VOB access faster and/or to increase the number of concurrent users without degrading performance.

- **Disk capacity**—Adequate disk storage is also very important: a VOB database must fit in a single disk partition, and VOB databases tend to grow significantly as development proceeds and projects mature. We recommend a disk capacity of at least **2Gb**.
- **Processing power**—The recommended speed for a VOB host's processor is **20–35 MIPS**. Make the most of the CPU cycles by keeping "private" processes—ClearCase client tools and views—off the VOB host.
- **Availability**—A VOB intended for shared access must be located in a disk partition that can be mounted by all ClearCase client hosts. Wherever possible, select a host that can be accessed with the same hostname by all ClearCase hosts. (A host with multiple network interfaces presents a different name through each interface.) If a VOB host has multiple names, you will need to create multiple *network regions*, to logically partition the network.

Planning for One or More VOBs

A VOB host that meets the description in the preceding section has this (very approximate) overall capacity:

- 2500 source file elements, each with a version tree that contains many versions
- 750 files that are build targets; the VOB typically stores multiple instances of each target (that is, multiple derived objects built at the same pathname), representing various build configurations in current use
- ability to service requests from 20 concurrent users on ClearCase client hosts around the network

It is up to your organization to decide how to allocate data to one or more VOBs on a VOB host. Here are the principal tradeoffs:

- Splitting data into several small VOBs greatly increases your flexibility: it is easy to move an entire VOB to another host; it is difficult to split a VOB into two parts and move one of them to another host.
- Typically, having multiple small VOBs makes for fewer performance bottlenecks than having one large VOB.

- Having fewer VOBs facilitates data backup.
- Having fewer VOBs facilitates synchronizing label, branch, and other definitions across all the VOBs.

In your VOB planning, keep in mind that you can make several distinct VOBs *appear* to be a single directory tree, using VOB symbolic links (Figure 6-1).

Note: Be sure that the text of a VOB symbolic link is a relative pathname, not a full pathname: ♦

```
% ls -l /vobs/project/lib          (wrong)
/vobs/project/lib -> /vobs/lib
% ls -l /vobs/project/lib          (right)
/vobs/project/lib -> ../lib
```

This ensures the link will be traversed correctly in all view contexts. See the *pathnames_ccase* manual page for more on this topic.

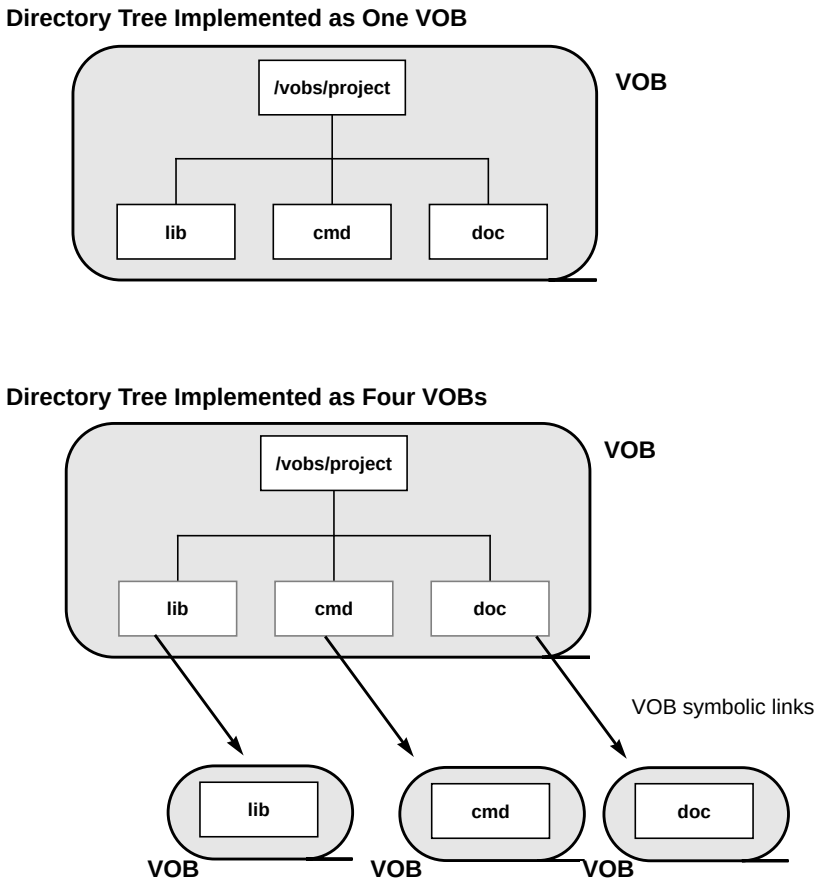


Figure 6-1 Linking Multiple VOBs Into a Single Directory Tree

Planning for Release VOBs

VOBs are not just for source files—you can also use them to store product releases (binaries, configuration files, bitmaps, and so on). Such VOBs tend to grow quickly; we recommend that in a multiple-architecture environment, releases for different platforms be stored in separate VOBs.

Modifying a VOB Host for ClearCase

Each VOB is managed by a battery of server processes, which run on the host where the VOB storage directory resides. These servers make significant demands on system resources. Accordingly, you should make sure that the system's configuration is adequate for ClearCase's needs.

Kernel Resources

You may need to adjust the following kernel resources on a VOB host:

- **Overall process table**—The operating system's process table should support **96** or more concurrent user processes. If more than three or four VOBs are to reside on the host, increase the size of the process table to at least **128**.
- **Overall file descriptor table**—The size of the operating system's file descriptor table should be at least **600**. If more than three or four VOBs are to reside on the host, increase the size of the file descriptor table further.

You may also find it beneficial to adjust a VOB hosts' kernel resources after ClearCase has been up and running for some time. For more on this topic, see "Manipulate Block Buffer Caches" on page 132.

Optional Software Packages

In order to ensure correct ClearCase operation, you may need to install one or more optional software packages available from your hardware vendor. Consult the installation instructions in the *CASEVision/ClearCase Release Notes* for more information.

Creating a New VOB

Follow these steps to plan and execute the creation of each new shared VOB:

1. **Log in to the VOB host**—Log in to the host you've selected to be the VOB host. As discussed in "Network-Wide ClearCase Administrator" on page 39, we suggest that you log in as *vobadm* (or some other "ClearCase administrator" identity).

2. **Choose a location for the VOB storage directory**—Make sure that the location is in a disk partition that has plenty of room for VOB database growth. This partition must be visible (mounted) on all ClearCase client hosts that will need to access the VOB. For example:

   ```
   /vobstore/flex.vbs
   ```

   In this example, */vobstore* could be a separate disk partition mounted on an empty subdirectory of "/". Just make sure you *know* where the disk storage is really located, and that it is globally visible.

3. **Choose a VOB-tag**—Each ClearCase client host mounts the VOB as a file system of type MVFS. Unless there is a compelling reason (perhaps you are trying to drive yourself crazy), all clients should mount the VOB at the same pathname. For example:

   ```
   /vobs/flex
   ```

   The full pathname of the VOB mount point is called its *VOB-tag*.

   Note: VOB-tags and view-tags are different in form: a VOB-tag is a full pathname; a view-tag is a simple directory name. ♦

4. **Create the VOB storage directory**—Using *cleartool* or *xclearcase*, enter a "create new VOB" command. Continuing the example from the preceding steps, the *cleartool* command would be:

   ```
   % cleartool mkvob -public -tag /vobs/flex
   /vobstore/flex.vbs
   ```

   You'll be prompted to enter a comment, which will be stored in an event record (`create versioned object base`) in the new VOB's database. You'll also be prompted to enter a password, because you're

creating a *public* VOB. *mkvob* validates your entry using the contents of the *VOB-tag password* file—*/usr/adm/atria/rgy/vob_tag.sec* on the network's *registry server host*.

Making the VOB public has two effects:

- On each client host, the command `cleartool mount -all` will be invoked by the ClearCase startup script to activate this VOB (and all other public VOBs).
- If the VOB becomes unmounted for any reason, any user (not just *root* or the VOB's creator) will be able to remount it.

After the VOB is created, *mkvob* reports the new VOB's registry and access-control information:

```
Host-local path: ccsvr01:/vobstore/flex.vbs
Global path:     /net/ccsvr01/vobstore/flex.vbs

VOB ownership:
  owner vobadm
  group dvt
Additional groups:
  group libdvt
```

In many cases, you've now completed the VOB-creation process. The following sections describe special cases and optional adjustments you may wish to make to the new VOB.

Adjusting the VOB's Identity Information

This section discusses changes that you may need to make to a new VOB's identity information. We discuss both the no-change-required and the change-required situations.

There are access-control issues to be addressed if (1) the prospective users of the VOB do not all belong to the same group, and (2) non-group members are to be prohibited from accessing data. (This issue was discussed in "VOBs and Views: Owner and Groups" on page 39.)

Case 1: One Group for All VOBs, Views, and Users

Small development organizations, and ones in which data security is not a major issue, sometimes place all users in the same group (for example, *dvt*). In such organizations, all ClearCase data structures should also belong to the common group—a VOB or view belongs to the principal group of its creator—and will be fully accessible to all users.

The example commands in Steps #1–#4 in "Creating a New VOB" on page 62 are sufficient to create a VOB in such a situation.

Case 2: Accommodating Multiple User Groups

If your organization has multiple user groups, there are several questions to be answered when you create a new VOB:

- **Who should be granted "write" access?**—Determine which users will be doing development work in the VOB, and compile a list of their principal groups. All these groups must be added to the VOB's group list. (If a user group is also the VOB's principal group, it need not be added to the group list.)
- **Should all others be granted "read" access?**—If so, then the VOB's mode should grant *others* (as opposed to *user* and *group*) "read" access. But if some users are to be restricted even from examining a VOB's data, then:
  - Make sure that the access mode of the VOB's root directory (top-level directory element) grants *others* no access rights at all.
  - Make sure that if a user is to be prohibited from accessing the VOB, his or her list of groups does not overlap the VOB's list of groups at all.

Example: Multiple Groups

Let's revisit the example in "Creating a New VOB" on page 62. The VOB created in Step #4 is owned by *vobadm*, with principal group *dvt*, and with *libdvt* as its only additional group. Suppose that the VOB will be used by ten developers, whose principal groups include *dvt*, *libdvt*, *exper*, and *visitor*.

1. **Add users' principal groups to the VOB's group list**—In this example, the groups *exper* and *visitor* are to be added to the VOB's group list. Only the *root* user can use the *protectvob* command.

```
% su vobadm
Password: <enter password>
# cleartool protectvob -add_group exper,visitor \
    /vobstore/flex.vbs
... <confirmation prompts and messages> ...
VOB ownership:
  owner vobadm
  group dvt
Additional groups:
  group libdvt
  group exper
  group visitor
# exit
%
```

2. **Remove 'other' access to the VOB**—To prevent users who do not belong to any of the VOB's groups from accessing the VOB, change the access mode of the VOB's root directory. This requires that the VOB be mounted, so make the change on a ClearCase client host.

```
% rlogin neptune -l vobadm
Password: <enter password>
% cleartool mount /vobs/flex                (just in case)
% cleartool protect -chmod o-rx /vobs/flex
Changed protection on "/vobs/flex".
% ls -ld /vobs/flex
drwxrwx---    3 vobadm      30 Jan 31 13:24 /vobs/flex
```

Ensuring the VOB's Global Accessibility

The output from the *mkvob* command (Step #4 above) includes a "global" (network-wide) pathname for the VOB storage directory:

```
Global path:     /net/ccsvr01/vobstore/flex.vbs
```

This pathname is heuristically derived—that is, it's an intelligent guess. Depending on the accuracy of the guess, you may have some more work to do in guaranteeing the VOB's accessibility to all users on all ClearCase hosts.

Case 1: Heuristic Guess Was Right

If all ClearCase client hosts can access the VOB at the "global pathname" reported by *mkvob*, you have no more work to do.

Case 2: Guess Was Wrong, But Global Pathname Does Exist

It might be the case that there *is* a global pathname, which all ClearCase client hosts can use to access the VOB storage directory, but *mkvob*'s heuristically-derived pathname is not the right one. The most common reasons are:

- use of a nonstandard auto-mount program
- use of a home-grown (perhaps manual) scheme for mounting file systems around the network

In this case, use the *mktag* command to correct *mkvob*'s guess. For example:

```
% cleartool mktag -replace -vob -public -tag /vobs/flex \
    -host ccsvr01 \
    -hpath /vobstore/flex.vbs \
          (the above information must be valid on the VOB host)
    -gpath /allvobs/flex.vbs \
          (valid pathname to VOB on all hosts)
    /vobstore/flex.vbs
          (valid pathname on the local host)
Vob tag registry password: <enter password>
  .
  .
```

Case 3: Global Pathname Does Not Exist

It may be that there is *no* global pathname for the VOB storage directory. The most common reason is that the VOB host has multiple network interfaces (is *multihomed*)—some client hosts might know it as *ccsvr01*, while others know it as *ccsvr01-gw*.

In this case, you must partition your network into multiple *network regions*; in each region, the global-pathname criterion must hold true. For background information, see "Network Regions" on page 28.

Continuing the example, suppose that the VOB host is named *ccsvr01* in region "uno", and *ccsvr01-gw* in region "dos". The *mkvob* command (Step #4) created the VOB-tag in one of the regions—it doesn't matter which one, because you should now use *mktag* to update/create public VOB-tags in *all* network regions:

```
% cleartool mktag -replace -vob -public -region uno \
    -tag /vobs/flex \
    -host ccsvr01 \
    -hpath /vobstore/flex.vbs \
    -gpath /net/ccsvr01/vobstore/flex.vbs \
        (valid pathname to VOB on all hosts in network region "uno")
    /vobstore/flex.vbs
Vob tag registry password: <enter password>
  .
  .
% cleartool mktag -replace -vob -public -region dos \
    -tag /vobs/flex \
    -host ccsvr01-gw \
    -hpath /vobstore/flex.vbs \
        (valid pathname to VOB on all hosts in network region "dos")
    -gpath /net/ccsvr01-gw/vobstore/flex.vbs \
    /vobstore/flex.vbs
Vob tag registry password: <enter password>
  .
  .
```

Note: As in Case 2, your `-gpath` value may need to take into account usage of a nonstandard auto-mount program or other mounting idiosyncrasies within each network region. ♦

For additional registry-related procedures, see Chapter 18, "Adjusting ClearCase Registry Information".

Creating Remote Storage Pools

Typically, you won't add remote storage pools to a VOB until a disk-space crisis occurs. But as you gain more experience with how ClearCase is used by your group and how VOBs grow, you may wish to add remote storage pools when you first create a VOB. This can help to postpone the disk-space crisis—perhaps even eliminate it altogether.

See “Creating Additional VOB Storage Pools” on page 113 for a step-by-step procedure.

Coordinating the New VOB with Existing VOBs

A typical project involves multiple VOBs. To ensure that they all work together, you will probably need to coordinate the VOBs’ *type* objects: branch types, label types, and so on. (See the *ClearCase Concepts Manual* for an introduction to type objects.)

For example, the following config spec might be used to create a view that selects the source files that went into an old release:

```
element * RELEASE_3
```

In order for this strategy to succeed, all relevant VOBs must define version label *RELEASE_3*—that is, label type *RELEASE_3* must be created in each VOB.

There is no single command that copies type objects from one VOB to another. You may be able to automate this process somewhat with a script—for example, using the output of an `lstype -brtype -long` command in one VOB to form a series of `mkbrtype` commands for use in another VOB.

Populating a VOB with Data

The new VOB is now ready to be populated with data by the development group. At this point, the VOB contains a single directory element, the VOB’s *root directory*. When the VOB is activated, this directory appears at the VOB-tag pathname—that is, at the VOB mount point.

Note: The root directory also contains a *lost+found* directory. See the *mkvob* manual page for a discussion of this directory. ♦

To users, a VOB appears to be a single UNIX directory tree. Thus, it makes sense to consider how your sources “naturally” fall into logically separate trees, and create one VOB for each one. If two projects do not share source files, place the sources in different VOBs. Typically, several or all projects

share some header (*.h*) files. Isolate these shared sources (for example, a */vobs/project/include* directory tree) in their own VOB.

If this approach concentrates too many elements in a single VOB, it can produce a performance bottleneck in accessing the VOB database, causing the VOB host to become "CPU-bound". Adding users to a project also increases the load on the VOB databases that they access and, thus, can also produce "CPU-bound" problems.

Example: Importing RCS Data

To illustrate migration of sources to ClearCase, we present a simple conversion scenario: using an entire RCS source tree to populate a newly-created ClearCase VOB. Suppose that the root of the RCS tree is */usr/libpub*, located on a host where the empty VOB has already been activated, at */proj/libpub*.

Creating the Conversion Scripts

1. **Go to the source data**—Change to the root directory of the existing RCS source tree:

   ```
   % cd /usr/libpub
   ```

2. **Run the conversion utility**—Use the RCS-to-ClearCase import utility, *clearcvt_rcs*, to create the scripts that will convert RCS files (*,v* files) to ClearCase elements:

   ```
   % clearcvt_rcs
   Converting element "./Makefile,v" ...
   Extracting element history ...
   .
   Completed.
   Converting element ...
   Creating element ...
   Element "./Makefile" completed.
     .
     .
   Element "./lineseq.c" completed.
   Creating script file cvt_dir/cvt_script ...
   ```

A "master conversion script" is created in subdirectory *cvt_dir* of the current working directory; the script's full pathname is */usr/libpub/cvt_dir/cvt_script*.

Running the Conversion Scripts

3. **Set a view configured with the default config spec**—You can use the *catcs* command to determine whether your current view (if any) has the default configuration. If not, set another view:

   ```
   % cleartool setview dft
   ```

4. **Go to the target VOB**—Change to the root directory of the newly-created *libpub* VOB—that is, the directory specified by its *VOB-tag*:

   ```
   % cd /proj/libpub
   ```

5. **Run the conversion script**—Invoke the "master conversion script" to populate the *libpub* VOB:

   ```
   % /usr/libpub/cvt_dir/cvt_script
   Converting files from /usr/libpub to .
   You are using the default config_spec
   Created element "././Makefile" (type "text_file").
   Changed protection on "././Makefile".

   Making version of ././Makefile

   Checked out "././Makefile" from version "/main/0".
   Comment for all listed objects:
   Checked in "././Makefile" version "/main/1".
     .
     .
   ```

   Note that there is no need to checkout or checkin the VOB's root directory element—this is handled automatically. If problems occur that cause the conversion script to terminate prematurely, you can simply fix the problem and restart the script.

Chapter 7

Setting Up ClearCase Views

This chapter discusses setting up of views for individual users, views to be shared by groups of users, and views through which VOBs will be made available to non-ClearCase hosts.

Setting Up an Individual User's View

In a typical ClearCase development environment, most views are created by individual developers, on their individual workstations, for their personal use. This model conforms well to ClearCase's client-server architecture, and takes advantage of *scalability*: as new users join the environment, they bring with them the processing power and disk storage of additional workstations. If a user's workstation has local storage, it makes sense for the user's view(s) to reside within his or her home directory. Alternatively, you can place the storage for some or all views on a central, well-backed-up file server host.

In deciding where to place views, keep in mind these architectural constraints:

- Each view has an associated server process, its *view_server*, which executes on the host where the view's storage directory is created.
- ClearCase must be installed on the host where a view storage directory is created and the *view_server* process runs.
- If a host is to keep several views (and their several *view_server* processes) active concurrently, it should be configured with extra main memory.

View Storage Requirements

Each ClearCase view is implemented as a *view storage directory*, a directory tree that holds a small database, along with a private storage area that contains view-private files, checked-out versions of elements, and unshared derived objects.

View Database

The view database is a set of UNIX files, located in subdirectory *db* of the view storage directory. Typically, this database is quite small (less than 1Mb), and presents no significant disk space problems.

View's Private Storage Area

A view's private storage area is implemented as a directory tree named *.s* in the view storage directory. By default, *.s* is an actual subdirectory, so that all data stored in the view will occupy a single disk partition.

If you anticipate that a view will need a great deal of private storage, you can use the `mkview -ln` command to create *.s* as a symbolic link, pointing to a location in another disk partition, perhaps on another host:

```
% cleartool mkview -tag david -ln /net/sirius/viewstore/1 ~/my.vws
```

In making this decision, consider that unshared derived objects typically make the greatest storage demand on a view. To obtain a useful estimate of the maximum disk space required for a view, calculate the total size of all the binaries, libraries, and executables for the largest software system to be built in that view. If several ports (or other variants) of a software system will be built in the same view, it must be able to accommodate the several kinds of binaries.

Setting Up a Shared View

Views can be shared by multiple users. For example, a project might designate a shared view in which all of its software components are built in preparation for a release. The entire application might be built each night in such a view.

An ideal shared view is located on a dedicated host that is configured similarly to a client workstation. If no dedicated host is available, distribute shared views around the network on the least-heavily-used (or most richly configured) client workstations. Avoid placing too many views on any single machine; avoid placing shared views on VOB hosts (but see the next section for an exception).

Here is a simple procedure for setting up a shared view:

1. **Determine who will be using the view**—In particular, determine whether all of the view's prospective users belong to the same group.
2. **(if necessary) Change your group**—If all of the view's prospective users belong to the same group, make sure that you are logged in as a member of that group. You may need to use a *newgrp*(1) command to switch your group.
3. **Set your umask appropriately**—A view's accessibility is determined by the *umask*(1) of its creator. If the view's users are all members of the same group, temporarily set your umask to allow "write by group members":

   ```
   % umask 2
   ```

   Otherwise, you must set your umask to allow any user write access:

   ```
   % umask 0
   ```

4. **Determine a location for view storage directory**—Use the discussion in "View Storage Requirements" on page 72 to decide whether the view's private storage area should be local or remote.
5. **Choose a view-tag**—Select a name that indicates the nature of the work that will be performed in the view. For example, you might select *integ_r1.3* as the tag for a view that will be used to produce Release 1.3 of your application.

6. **Create the view storage directory**—Enter a "create new view" command:

```
% cleartool mkview -tag integ_r1.3 \
    /net/ccsvr05/viewstore/integr13.vws
Created view.
Host-local path: ccsvr05:/viewstore/integr13.vws
Global path:     /net/ccsvr05/viewstore/integr13.vws
It has the following rights:
User : vobadm   : rwx
Group: dvt      : rwx
Other:          : r-x
```

   (See "View's Private Storage Area" on page 72 for a command that creates a view with a remote private storage area.)

7. **Verify your work**—Examine the *mkview* command's output to verify that the access permissions are in accordance with your decisions in Steps #1–#3. In addition, examine the "host-local path" and "global path". You may need to make adjustments similar to those discussed in "Ensuring the VOB's Global Accessibility" on page 65.

8. **Restore your original umask and/or group**—Enter a *umask* command to restore your original umask setting; or just exit the shell process. Exiting the shell is also the easiest course to take if you've changed your group setting with a *newgrp* command.

Setting Up an Export View for Non-ClearCase Access

A ClearCase VOB can be made available to hosts on which ClearCase is not installed. This *non-ClearCase access* feature involves setting up an *export view*, through which the VOB will be seen on the non-ClearCase host (Figure 7-1):

1. A ClearCase client host—one whose kernel includes the MVFS—activates (mounts) the VOB.

2. The host starts an export view, through which the VOB will be accessed by non-ClearCase hosts.

3. The host uses a ClearCase-specific exports file to export a view-extended pathname to the VOB mount point—for example, */view/exp_vu/vobs/proj*.

4. One or more non-ClearCase hosts in the network perform an NFS mount of the exported pathname.

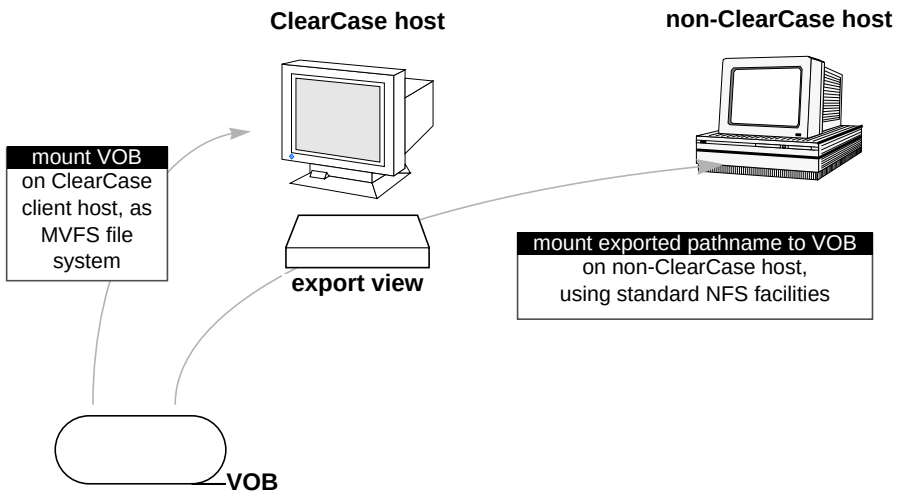


Figure 7-1 Export View for Non-ClearCase Access

The *exports_ccase* manual page describes the simplest (and recommended) setup, in which the VOB and the export view are located on the same host. The following sections discuss this issue in greater detail, including advice on how to proceed if you don't wish to co-locate the VOB and export view.

Exporting Multiple VOBs

If you adopt the recommendation to co-locate VOBs and their export views, it is likely that developers working on a non-ClearCase host will access several export views at the same time. For example, a project might involve three VOBs located on three different hosts. Since each VOB and its export view are located on the same host, three different export views are involved. On the non-ClearCase host, the NFS mount entries might be:

```
saturn:/view/beta/vobs/proj /vobs/proj nfs rw,hard 0 0
neptune:/view/exp_vu/vobs/proj_aux /vobs/proj_aux nfs
rw,hard 0 0 pluto:/view/archive_vu/vstore/tools /vobs/tools
nfs rw,hard 0 0
```

The three VOBs can be accessed on the non-ClearCase host as subdirectories of */vobs*. But developers must keep in mind that three views are involved, for such operations as checkouts. Developers need not be concerned with multiple-view issues when building software on the non-ClearCase host.

Multihop Export Configurations

In a *non-ClearCase access* situation, a single data-access can involve three hosts:

- the host on which the VOB storage directory resides
- the host on which the storage directory of the export view resides
- the non-ClearCase host

This *multihop* situation is not supported in pure-NFS environments, but is made possible by MVFS-level communication between the two ClearCase hosts. But creating a multihop configuration introduces the possibility of "access cycles", in which two of the hosts depend on each other for network-related services, or such a dependency is created through "third-party" hosts. Such situations result in timeouts (if VOBs are soft-mounted) or deadlocks (if VOBs are hard-mounted).

A sure way to avoid access cycles is to avoid multihop configurations altogether, as described in the *exports_ccase* manual page:

- Locate the storage directory of the export view on the same host as the storage directory for the VOB.
- Make sure that neither the VOB nor view has remote data storage. That is, the VOB should not have any remote storage pools, and the view's private storage area (*.s* directory tree) must be an actual subdirectory, not a symbolic link to another host.

If you wish to use a multihop configuration, you must ensure that the *VOB host* (and its "pool hosts", if any) never request services from the *view host*. This ensures that no process on the VOB and pool hosts creates an access cycle with the view host. Figure 7-2 illustrates an access-layering scheme that avoids access cycles.

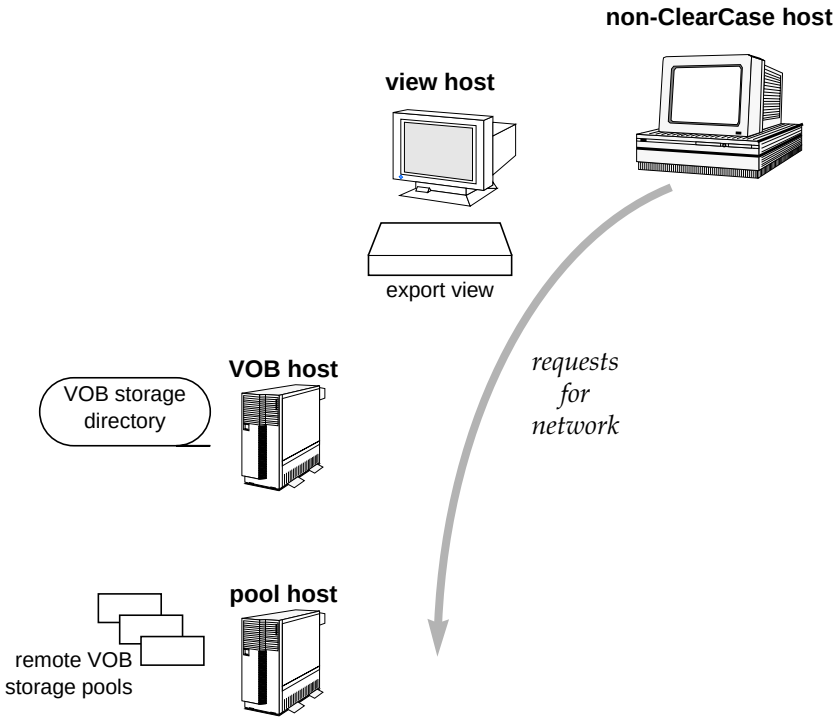


Figure 7-2 Avoiding Access Cycles in Non-ClearCase Access

In this scheme, higher-layer hosts always request services from lower-layer hosts. A request for *any* network service (not just ClearCase services) must never be made back to the view host, where the *view_server* for the export view runs, either directly or through some other host.

You might achieve the correct layering by never allowing any users to run processes on a host used for an export view, either directly or indirectly—no home directories, and no remote logins (except from non-ClearCase hosts). In addition, make sure that no over-the-network backups of the view server hosts are per- formed on the VOB server or pool hosts.

Restricting Exports to Particular Hosts

In a multihop situation, we recommend using an *-access* option in each entry in the ClearCase exports file, */etc/exports.mvfs*. This restricts the export to specified non-ClearCase host(s) and/or netgroups. This greatly reduces the likelihood of creating access cycles. For example:

```
/view/exp_vu/usr/src/proj -access=galileo:newton:bohr:pcgroup
```

When combining *-access* with other options, be sure to specify them all as a comma-separated list off a single hyphen.

Preventing Accidental Deletion of Data by *crontab* Entries

This chapter describes changes made automatically during ClearCase installation to ensure that *crontab*(1) scripts do not accidentally delete ClearCase data. In addition, we describe situations in which you may need to take measures to prevent such accidents from occurring.

Preventing Recursive Traversal of '/'

When the ClearCase MVFS is active on a client host, the UNIX file system is modified so that the root directory contains itself recursively (Figure 8-1). This makes certain "recursive" pathnames valid. For example:

```
/view/alpha/view/beta/view/alpha/view/gamma/usr/src/lib
```

This situation causes commands that traverse the entire directory tree, starting at the UNIX root directory (/), to loop infinitely. In particular, many UNIX systems configure the *root* user to have a daily *crontab*(1) script that performs a "cleanup" on the file system tree. If the script uses a *find* / command, it runs the risk of looping infinitely.

Note: This situation applies equally to commands and programs executed by any user, either interactively or through a script. ♦

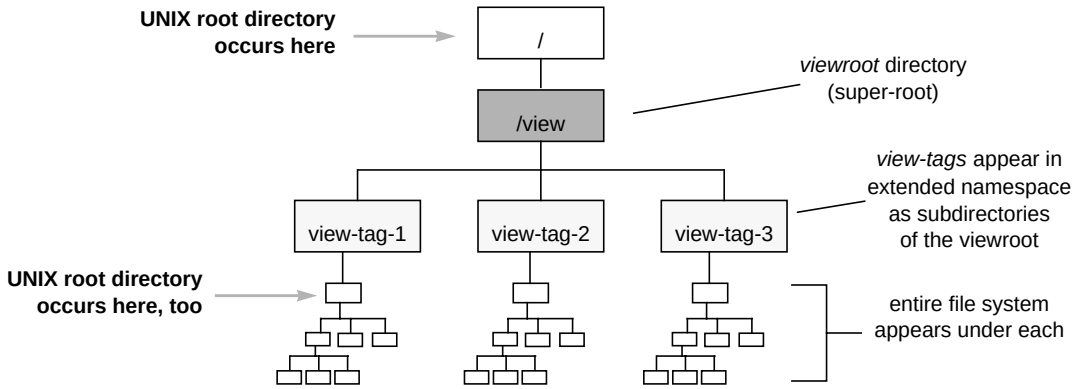


Figure 8-1 Viewroot Directory as a Super-Root

Crontab Modification During ClearCase Installation

The ClearCase installation script, *install_release*, analyzes the *crontab* file of the *root* user on a client host. It modifies entries in this file to prevent the recursive traversal problem (or displays a message warning that it cannot perform the modification).

After installation, you should verify the correctness of *install_release*'s changes. In addition, you should modify the *crontab* entries of other users, according to the instructions in the next section.

Modifying a Crontab Entry

Use the following procedure to analyze and, if necessary, modify all of a host's *crontab* entries.

1. **Analyze the crontab entries**—Determine what entries will encounter the recursion problem:
   ```
   % su
   Password: <enter root password>
   # grep "find /" /usr/spool/cron/crontabs/*
   /usr/spool/cron/crontabs/root:15 3 * * * find / -name .nfs\*
       -mtime +7 -exec rm -f {} \; -o -fstype nfs -prune
   ```
   (typical output)

In the example above, and throughout this section, long lines are broken for readability. In actual *crontab* files, each entry must be contained on a single physical text line.

2. **Revise crontab entries**—You must modify each *crontab* file in which an offending entry was found. For example:

```
# su - username                 (switch user identity)
# crontab -l > /tmp/C           (create temporary file with that user's crontab entries)
# vi /tmp/C                     (modify those entries)
# crontab < /tmp/C              (configure the modified entries)
```

   During the edit session, change the command containing *find /* by adding the host-appropriate options indicated by **boldface** below:

   - **SunOS host**—Since the viewroot directory (typically, */view*) is mounted as a file system of type *mvfs*, use the *–prune* option to prevent recursion:

```
15 3 * * * find / -name .nfs\* -mtime +7
    -exec rm -f {} ';' -o -fstype nfs -prune
    -o -fstype mvfs -prune
```

   - **HP-UX host**—No *crontab* entry is provided by default. The following entry searches file systems of type *hfs* and *cdfs* only (the viewroot directory is neither of these):

```
15 3 * * * find / -path /view -prune
    -o -name .nfs\* -mtime +7 -exec rm -f {} \;
    -o -fstype nfs -prune
```

   - **IRIX host**—Make sure that the *-local* option is present, to prevent crossing of mount points. Since the viewroot directory is mounted, this prevents recursion:

```
0 5 * * * find / -local -type f
    '(' -name core -o -name dead.letter ')'
    -atime +7 -mtime +7 -exec rm -f '{}' ';'
```

   - **OSF/1 host**—Make sure that the *-xdev* option is present, to prevent crossing of mount points. Since the viewroot directory is mounted, this prevents recursion:

```
15 3 * * * find filesys-1 filesys-2 ... -xdev
    -mtime +7 -exec rm -f {} \;
```

Preventing Accidental Deletion of the Lock Manager Socket

When it begins execution, the *lockmgr* program creates a socket, */tmp/.A/almd*, through which it communicates with local and remote calling processes. To reduce the likelihood of accidental deletion, the socket is created within a subdirectory of */tmp*, and is owned by the *root* user.

On each ClearCase host, you should examine the *root* user's *crontab* file, modifying it if necessary to ensure that the *lockmgr* socket is not deleted accidentally. For example, the following *find* command includes a "not a socket" clause:

```
find /tmp ! \( -type s \) -exec rm -f {} \;
```

Chapter 9

Data Backup: VOBs and Views

The most important maintenance task for a system administrator is ensuring frequent, reliable backups of essential disk storage. Because of this task's preeminent importance, we devote this chapter to it. The next chapter discusses other regular maintenance tasks.

Backup Tools

ClearCase does not include any data backup tools. All ClearCase data is stored in standard files, within standard directory trees; thus, you can use any backup tools at your disposal. The standard UNIX utilities *tar*(1) and *cpio*(1) are well-suited to backing up ClearCase data structures.

Backing Up a VOB

By default, a VOB storage directory is wholly contained in a single directory tree, which resides in a single disk partition. If you use a file-oriented backup tool, you need only specify the VOB storage directory to ensure a complete backup. If you use a disk-partition-oriented backup tool, you need only specify the partition name.

Note: The commands listed in the following sections don't require a ClearCase view context. The commands "don't care" whether or not a VOB is activated (mounted) on client hosts. ♦

Determining a VOB's Location

To determine the location of a VOB's storage directory, use *lsvob*:

```
% cleartool lsvob -long /vobs/flex
VOB on host: ccsvr01
VOB server access path: /vobstore/flex.vbs
     (use this information if your backup program works locally)
 .
 .
  Tag: /vobs/flex
    Global path: /net/ccsvr01/vobstore/flex.vbs
     .
     .
    Region: uno
         (use this information if your backup program runs over the network)
```

Specify the appropriate pathname to your backup program. With a program that backs up entire disk partitions, you'll need to determine what partition the VOB storage directory resides in.

Ensuring a Consistent Backup

A backup will represent a self-consistent snapshot of a VOB storage directory's contents only if the VOB remains unmodified while the backup program is working. You need not unmount a VOB to ensure this—just lock it before backing it up, and unlock it afterward.

```
% cleartool lock -vob /net/ccsvr01/vobstore/flex.vbs
Locked versioned object base
"/net/ccsvr01/vobstore/flex.vbs".

   <perform backup>

% cleartool unlock -vob /net/ccsvr01/vobstore/flex.vbs
Unlocked versioned object base
"/net/ccsvr01/vobstore/flex.vbs".
```

Partial Backups

If you use a file-oriented backup program, you may wish to back up only some subdirectories within the VOB storage directory, in order to save time. Use the guidelines in Table 9-1 (carefully!) to determine the relative importance of the various components of a VOB:

Table 9-1 VOB Components for Partial Backups

VOB Component	Importance for Backup
top-level VOB storage directory	absolutely essential
VOB database subdirectory	absolutely essential
source storage pools	absolutely essential
derived object storage pools	important, but not essential
cleartext storage pools	optional

Backing up derived object storage pools is not crucial because, by definition, DOs can be rebuilt from sources. The importance of backing up these pools may change over time:

- In the early stages of a project, when the source base in changing rapidly, the useful life of most derived objects is very short. Omitting DO storage pools from a backup regimen probably won't be much noticed by developers.
- When a project is relatively stable, a VOB's DO storage pools contain many often-reused objects. At this stage, a complete build of a software system might *wink-in* virtually all DOs, rather than building them. Loss of a DO storage pool might increase the time requirement for such a complete system build by an order of magnitude.

Caution: A shared derived object has two parts: an object in the VOB database and a data container in a DO storage pool. Loss of DO data containers (for example, through failure to back them up) throws the VOB's database "out of sync" with its DO storage pools. To resynchronize, you must delete all the "dataless" DOs from the VOB database, using *rmdo*.

Backing up cleartext storage pools is not important at all, because they are merely caches that enhance performance. ClearCase type managers recreate cleartext data containers automatically, as necessary.

Example of Partial VOB Backup

Suppose you decide to back up all but the cleartext storage pools of the VOB located at */vobstore/flex.vbs*. This would entail:

- Backing up the files, but not the subdirectories of */vobstore/flex.vbs*.
- Backing up the entire contents of directory */vobstore/flex.vbs/.identity*, which contains the VOB's ownership and group membership information.
- Backing up the entire directory tree at */vobstore/flex.vbs/db*, which contains the VOB database.
- Backing up the entire directory tree at */vobstore/flex.vbs/s*, which contains the VOB's source storage pools.
- Backing up the entire directory tree at */vobstore/flex.vbs/d*, which contains the VOB's derived object storage pools.

Backing Up a VOB with Remote Storage Pools

If a VOB has remote storage pools, then its on-disk storage is probably not wholly contained within a single disk partition. It is quite likely that the storage is distributed among two or more hosts. In this situation, the overall procedure is:

1. Lock the VOB.
2. Back up the VOB storage directory.
3. Back up some or all of the VOB's storage pools.
4. Unlock the VOB.

For Step #3, use the *lspool* command in any view to determine which storage pools are remote, and their actual locations:

```
% cleartool lspool -vob /vobs/flex
13-Jan.16:58   vobadm     pool "cdft"
  "Predefined pool used to store cleartext versions."
26-Jan.22:02   vobadm     pool "cltxt01"
  "remote cleartext storage pool for 'flex' VOB"
```

(only remote storage pool)

```
13-Jan.16:58   vobadm     pool "ddft"
  "Predefined pool used to store derived objects."
13-Jan.16:58   vobadm     pool "sdft"
  "Predefined pool used to store versions."
% c lspool -long -vob /vobs/flex cltxt01 pool "cltxt01"
     .
     .
  pool storage global pathname
   "/vobstore/flex.vbs/c/cltxt01"
     .
     .
```

If you are not sure which storage pools are remote, enter a *lspool –long* command to list the "pool storage global pathname" of every pool, and examine these pathnames to determine which ones specify remote locations.

Restoring a VOB from Backup

The following procedure restores a VOB backup without disrupting ongoing work. We assume that the VOB is the same one discussed in "Backing Up a VOB" on page 83.

Note: You can restore a VOB to a new location, on the same host or on another host. In this case, you must reregister the VOB at its new location (Step #8). ♦

1. **Unmount the VOB**—The VOB must be unmounted on each client host. For example:

   ```
   % cleartool umount /vobs/flex
   ```

2. **`Go to the VOB host`**—Log in, as the *root* user, to the host where the VOB storage directory resides:

   ```
   % rlogin ccsvr01 -l root
   Password: <enter root password>
   #
   ```

3. **Check disk space availability**—Make sure there is enough free space in the VOB's disk partition to load the backup copy:

   ```
   # cleartool space /vobstore/flex.vbs
     Use(Mb)  %Use  Directory
        27.0    2%  VOB database
          /net/ccsvr01/vobstore/flex.vbs
        33.0    3%  cleartext pool
          /net/ccsvr01/vobstore/flex.vbs/c/cdft
      .
      .
   --------  ----  ---------------------------------------
      312.9   28%  Subtotal
      828.4   74%  Filesystem /net/ccsvr01/vobstore
   (capacity 1115.1 Mb)
   ```

   If the available space is insufficient, delete the VOB storage directory, or use other means to make enough space available.

4. **Shut down ClearCase on this host**—This ensures that ClearCase processes associated with the VOB are terminated. For example:

   ```
   # /etc/rc.atria stop
   ```

   Note: The name of the ClearCase shutdown script varies from system to system. See the init_ccase manual page. ♦

5. **Move the original VOB aside**—If it still exists, rename the VOB storage directory:

   ```
   # mv /vobstore/flex.vbs /vobstore/flex.OLD
   ```

6. **Restart ClearCase on this host**—This reenables use of any other VOBs located there. For example:

   ```
   # /etc/rc.atria start
   ```

7. **Load the backup**—Restore the VOB storage directory from the backup medium. For example:

```
# mkdir /vobstore/flex.vbs
# cd /vobstore/flex.vbs
# tar -xvp
```

Note: Each VOB storage area includes a directory named *.identity*, which stores files with special permissions: the *setUID* bit is set on file *uid*; the *setGID* bit is set on file *gid*. You must preserve these special permissions when you restore a VOB backup:

- If you used *tar*(1) to back up the VOB, use *tar*'s *-p* option when restoring the VOB. In addition, make sure to enter the *tar* command as the VOB owner or as the *root* user.
- If you used *cpio*(1) to back up the VOB, no special options are required in the *cpio* command that restores the backup data.

If the VOB has remote storage pools, restore the backups of these pools at this point. ♦

8. **Reregister the VOB**—This is necessary only if you restored the VOB to a new location. For example, if you restored the VOB to new location */vobst_aux/flex.vbs*:

```
# cleartool unregister -vob /vobst_aux/flex.vbs
# cleartool register -vob /vobst_aux/flex.vbs
    (Yes, unregister followed by register)
# cleartool mktag -vob -replace -tag /vobs/flex
/vobst_aux/flex.vbs
```

9. **Mount the restored VOB on one host**—If the VOB host is also a client host, activate it there. Otherwise, use some other ClearCase host:

```
# cleartool mount /vobs/flex
```

10. **Unlock the restored VOB**—The VOB should have been locked before it was backed up. (See "Ensuring a Consistent Backup" on page 84.)

```
# cleartool unlock -vob /vobstore/flex.vbs
Unlocked versioned object base "/vobstore/flex.vbs".
```

11. **Mount the restored VOB on all hosts**—The restored copy of the VOB is now ready to use. Have users remount the VOB on their workstations, using `cleartool mount`.

Reestablishing Consistency of a View's "Derived Object State"

Note: This section applies in any situation where a VOB's database has been "rolled back" to a previous state. ♦

After you restore a VOB from backup, its VOB database may be out-of-date with respect to certain derived objects. The "old" database won't know about any DOs that were created in subsequent ClearCase builds. This will cause errors during hierarchical builds in which those "late-arriving" DOs are reused to construct higher-level targets:

```
======== Rebuilding "libbld.a"========
building libbld.a
        rm -f libbld.a
        rm -f /vobs/atria/sun5/pvtlib/libbld.a
        /opt/SUNWspro/bin/cc -c -o ...
        ar cq libbld.a  bld.o bld_pp.o ...

Will store derived object "/vobs/proj/sun5/libbld_V.o"
Will store derived object "/vobs/proj/sun5/libbld.a"
clearmake: Error: INTERNAL ERROR detected and logged in
"/var/adm/atria/error_log".
```

The *error_log* file shows:

```
Sunday 12/19/93 15:55:02. host "scandium", pid 440, user
"chase"
Internal Error detected in "../bldr_vob.c"line 114
clearmake/cm/bldr_vob:
Error: VOB "scandium:/vobs/proj"
missing config record for derived object (OID)
"0b5759d0.fb1811cc.a0af.08:00:69:02:2e:aa"
```

To reestablish the view's consistency with the VOB:

1. **Determine which DOs are causing the inconsistency**—The *cleartool ls* command annotates them with *[no config record]*:

   ```
   % cleartool ls
   bldr_comm.ugh@@09-Dec.18:26.287028
   bldr_cr.msg.o [no config record]
   bldr_cr.o [no config record]
   bldr_cr.ugh [no config record]
   ```

```
bldr_cr_cache.msg.c@@24-May.20:51.42929
  .
  .
```

2. **Remove the offending DOs**—Use the standard *rm*(1) command:

```
% rm bldr_cr.msg.o bldr_cr.o bldr_cr.ugh
```

Backing Up a View

Backing up views is similar to backing up VOBs, but is often simpler:

- Users may create views under their home directories. In this case, whatever backup regimen you have in place to back up users' home directories will automatically pick up their view storage directories, as well.
- Views do not have multiple storage pools, some of which may be local and others remote. A view has a single private storage area, its *.s* subdirectory, which can be either local or remote.
- It never makes sense to attempt a partial backup of a view storage directory—all the data is important, because it's the only copy of one or more users' current work.

Use the following procedure to back up a view.

1. **Determine the location of the view storage directory**—Use the *lsview* command:

```
% cleartool lsview akp_vu
View on host: neptune
View server access path: /home/akp/views/akp.vws
        (use this information if your backup program works locally)
 .
 .
  Tag: akp_vu
    Global path: /net/neptune/home/akp/views/akp.vws
 .
 .
    Region: dvt
        (use this information if your backup program runs over the network)
```

2. **Ensure self-consistency of the backup**—A view cannot be locked with the *lock* command. To guarantee that a view is inactive when it is backed up:
   - Warn ClearCase users that the view is about to become inactive.
   - Use *kill*(1) to terminate the associated *view_server* process on the host where the view storage directory resides.
   - A user's subsequent *setview* or *startview* command will invoke a new *view_server* process. If you are not confident that user's will refrain from such activity, rename the view storage directory. (In the following steps, we assume that you have not renamed it.)
3. **Determine whether the view has remote storage**—You can use a standard *ls*(1) command:
   ```
   % cd /home/akp/views/akp.vws
   % ls -ld .s
      ...  .s -> /net/ccsvr04/viewstore/akp.stg
   ```
   (symbolic link indicates remote private storage area)
4. **Enter the backup command(s)**—If the view's private storage area is local, then a single command can back up the entire view storage directory tree. Use the local address or the network-wide address listed by *lsview* in Step #1:

   ```
   % cd /home/akp/views
   % tar -cv akp.vws
   ```

   If the view's private storage area is remote, you need to perform a second backup (typically, using a different backup tape):

   ```
   % tar -cv /net/ccsvr04/viewstore/akp.stg
   ```
5. **Reactivate the view**—If you renamed the view storage directory in Step #2, restore it to its original name. Inform users that they may resume using the view, with *setview* and *startview* commands.

Restoring a View from Backup

Use the following procedure to restore a backup view storage directory. We assume that the view is the same one discussed in "Backing Up a View" on page 91.

Note: You can restore a view to a new location, on the same host or on another host. In this case, you must reregister the view at its new location (Step #5). ♦

1. **Go to the view host**—Log in to the host where the view storage directory resides:

   ```
   % rlogin neptune
   ```

2. **Check disk space availability**—Make sure there is enough free space in the view's disk partition to load the backup copy. If necessary, delete the view storage directory, or use other means to make enough space available.

3. **Move the original view aside**—If it still exists, rename or delete the view storage directory:

   ```
   # mv /home/akp/views/akp.vws /home/akp/views/akp.vws.OLD
   ```

4. **Load the backup**—Restore the view storage directory from the backup medium. For example:

   ```
   % cd /home/akp/views
   % tar -xv
   ```

   If the view's private storage area is remote, restore its backup, too.

5. **Reregister the view—**This is necessary only if you restored the view to a new location. For example, if you restored the view to new location */usr2/akp.vws*:

   ```
   # cleartool unregister -view /usr2/akp.vws
   # cleartool register -view /usr2/akp.vws
   ```
   (Yes, unregister followed by register)
   ```
   # cleartool mktag -view -replace -tag akp_vu /usr2/akp.vws
   ```

6. **Reactivate the view—**The restored copy of the view is now ready to use. Use a *setview* or *startview* command to start a *view_server* process. Inform the user(s) of the view that it is available again.

Periodic Maintenance of the Data Repository

This chapter discusses maintenance procedures, both automatic and manual, for ClearCase VOBs and views. Invoking these procedures periodically controls the growth of these data structures.

VOB Storage Maintenance

VOB administration involves a continual trade-off between these goals:

- Ensuring that all important data (and meta-data) is preserved.
- Discarding data (and meta-data) that is no longer important, in order to minimize disk-space requirements.

Figure 10-1 shows how VOB storage pools and VOB databases grow in regular usage; it also lists the maintenance commands ("scrubbers") that control growth of these storage areas.

When ClearCase is installed on a host, the *root* user's *crontab*(1) file is modified to include a daily maintenance procedure (*ccase_cron.day*) and a weekly maintenance procedure (*ccase_cron.wk*). These scripts, in turn, invoke other scripts that run the scrubber programs.

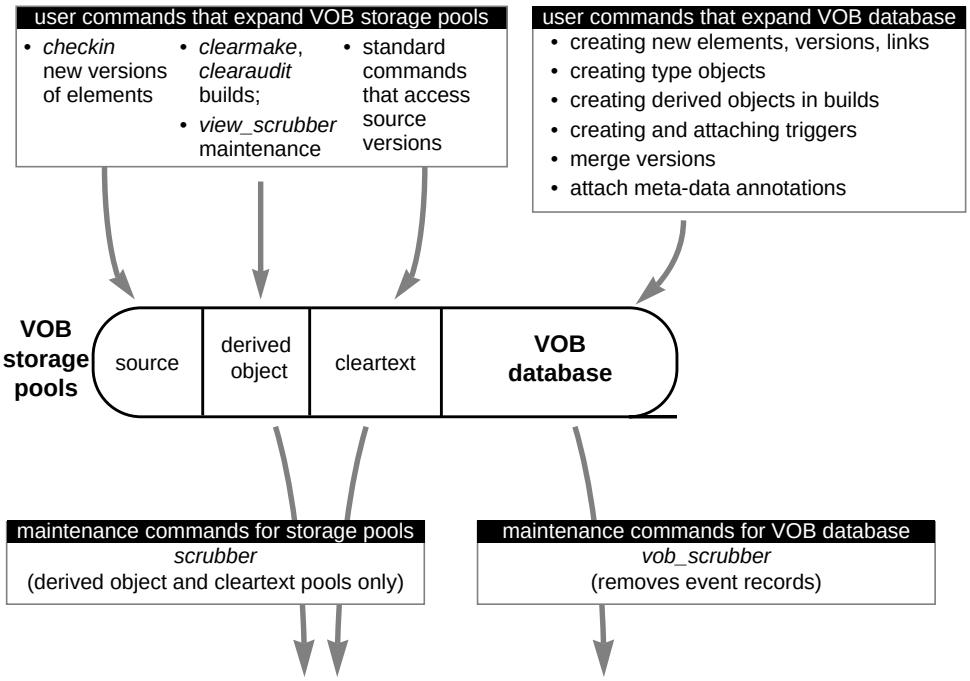


Figure 10-1 Controlling VOB Growth

Scrubbing VOB Storage Pools

A host's daily VOB-maintenance procedure scrubs the storage pools of all VOBs whose storage directories reside on that host:

- Source pools are never scrubbed automatically. Source versions are too valuable to be routinely deleted. (But see "Removing Unneeded Versions from a VOB" on page 108.)
- Derived object pools are scrubbed to delete DO data containers that are no longer being used by any view (those whose *reference counts* are zero). This also removes the corresponding derived objects from the VOB database.
- Cleartext pools are scrubbed to control their size. These pools are essentially caches; scrubbing all data containers in a cleartext pool would only affect ClearCase performance, not its correctness.

Scrubbing of storage pools is performed by the *scrubber* utility. Each derived object and cleartext storage pool has its own *scrubbing parameters*, which control how *scrubber* processes that pool. To change the way VOB storage pools are scrubbed on a host, you can:

- Change the invocation of the *scrubber* utility in script */usr/atria/config/cron/scrubber_day.sh*.
- Change the scrubbing parameters for an individual pool, using a `mkpool -update` command.

See "Adjusting Storage Pool Scrubbing" on page 117 for an example of modifying a pool's scrubbing parameters.

Scrubbing VOB Databases

Almost every change to a VOB is recorded in the VOB database as an *event record*. Some event records have permanent value, such as those for the creation of elements and versions. Others, however, may not be useful at all to your organization, or may lose their value as time passes. (For example, you probably don't care about the removal of an unneeded or obsolete version label.)

A host's weekly VOB-maintenance script invokes the *vob_scrubber* utility, removing unwanted event records from all VOBs whose storage directories reside on that host. Each host has its own configuration file, */usr/atria/config/vob/vob_scrubber_params*, which controls *vob_scrubber* operation; for even greater control, each VOB can have its own configuration file, with the same name. See the *vob_scrubber* manual page for more information.

Database Scrubbing: Logical vs. Physical

Deletion of event records and derived objects from a VOB database by the *vob_scrubber* and *scrubber* utilities is logical, rather than physical. That is, the scrubbers do not reduce the size of any file in the VOB database subdirectory (*db*). Instead, they increase the amount of "free space" within these files, for use by newly-created event records and derived objects.

If you need to actually shrink a VOB's on-disk storage, use the *reformatvob* command, which discards all such "free space". In general, however, we don't recommend routinely shrinking VOBs—it's usually sufficient to have maintenance procedures keep them from growing too fast.

View Storage Maintenance

Like any other isolated workspace, a view's private storage area tends to accumulate some "junk": core dump files, text-editor backup files, excerpts from mail messages and source files, and so on. These view-private files can take up a significant amount of disk space. As administrator, encourage users to clean up their own private views periodically; for shared views, the cleanup task may fall to you. See "Manual Cleanup of a View" on page 127.

Users can remove derived objects from their views using standard tools, such as the UNIX *rm*(1) command and *make clean* targets in *makefiles*.

From an administrator's standpoint, limiting the growth of a view storage directory typically involves just one issue: removing redundant derived object (DO) data containers. When a DO is first built by *clearmake* or *clearaudit*, its data container is placed in the private storage area of the user's view. The first time the DO is *winked-in* to another view, the data container is merely *copied*, not moved, to a VOB's derived object storage pool. (Moving it might wreak havoc with user processes that are currently accessing the DO.) This leaves a redundant copy of the data container in view-private storage.

Typically, you need not do *anything* about these redundant copies:

- In a view that is frequently used for builds, old (and potentially redundant) DO data containers are replaced by newer ones by the execution of build scripts.
- There can be at most one redundant copy of each DO in a view. (Contrast this with the situation for VOBs: if the *scrubber* utility is never invoked, the VOB will accumulate an ever-growing number of DOs that are no longer used.)

Unless disk storage is extremely scarce, these factors may make it not worth the effort to clean up redundant data containers in view-private storage.

Accordingly, ClearCase does not include any automated procedures for removing them.

Scrubbing View-Private Storage

If you decide that redundant DO data containers must be removed from some view's private storage area, use the *view_scrubber* utility. You can also use this utility to migrate the data containers of unshared (not yet *winked-in*) DOs to VOB storage—that is, to make the VOB(s) "pay the storage cost" instead of the view.

The following example shows how some DOs can be built, and then immediately transferred to VOB storage:

```
% clearmake hello
   <build messages>
% /usr/atria/etc/view_scrubber -p hello *.o
Promoted derived object "hello"
Scrubbed view-resident data container for "hello"
Promoted derived object "hello.o"
Scrubbed view-resident data container for "hello.o"
Promoted derived object "util.o"
Scrubbed view-resident data container for "util.o"
```

See the *view_scrubber* manual page for more information.

User-Supplied Maintenance Procedures

The daily and weekly ClearCase VOB-maintenance scripts automatically execute "local" supplementary scripts:

- */usr/atria/config/cron/ccase_local.day* is executed each day, after all standard maintenance procedures.
- */usr/atria/config/cron/ccase_local.wk* is executed once each week, after all standard maintenance procedures.

No error occurs if either of these user-supplied files does not exist. "Adjusting Storage Pool Scrubbing" on page 117 includes examples of supplementary maintenance scripts.

Caution: 'Local' Scripts May Not Really be Local

Depending on how ClearCase is installed, the directory */usr/atria/config/cron* may or may not actually be local to a particular host. If two or more hosts share the same */usr/atria/config/cron* directory, you may need to have the "local" script perform conditional processing, based on the hostname. Alternatively, you can use a completely separate mechanism to invoke supplementary maintenance procedures.

Chapter 11

Occasional VOB Maintenance

This chapter presents step-by-step procedures for a variety of VOB-maintenance tasks.

Moving a VOB (Same Architecture)

This section presents a procedure for moving a *VOB storage directory* to another location, either on the same host or on another host with the same architecture. (To move a VOB to a host of a different architecture, see "Moving a VOB (Different Architecture)" on page 104.) For clarity, we use an example:

- The current location of the VOB storage directory to be moved is */vobstore/libpub.vbs*, on a host named *sol*.
- The VOB is mounted by client hosts at */proj/libpub*.
- The new location for the VOB storage directory is */src_2/vobstore/libpub.vbs*. We consider two cases: (1) the new location is also on *sol*; (2) the new location is on another host, named *ccsvr04*.

To move the VOB, follow these steps:

1. **Determine whether the VOB has any nonlocal storage pools.**

```
% cleartool lspool -long -vob /proj/libpub | \
    egrep '(^pool|link)'
pool "cdft"
pool "ddft"
pool "sdft"
pool "s_2"
  (remote storage pool)
pool storage link target pathname "/net/ccsvr04/ccase_pools/s_2"
  (this pathname must be valid on new VOB host and on all client hosts)
```

2. **Verify the validity of pathnames to nonlocal pools**—Moving a VOB storage directory does not move any of its remote storage pools. You must make sure that the VOB's new host will access each remote storage pool using the same "global pathname" as the VOB's current host:

   - If you are moving the VOB to another location on the same host, the validity of remote storage pool global pathnames is assured.
   - If you are moving the VOB to a different host, log in to that host and verify that all the remote storage pool global pathnames are valid on that host.

3. **Deactivate the VOB**—On each host where it is currently active, this command deactivates it:

   ```
   % cleartool umount /proj/libpub
   ```

4. **Back up the VOB storage directory**—Use the procedure in "Backing Up a VOB" on page 83 (or in "Backing Up a VOB with Remote Storage Pools" on page 86).

5. **Lock the VOB**—Do this as the *root* user on the host where the VOB storage directory resides:

   ```
   % rlogin sol -l root
   Password: <enter root password>
   # cleartool lock -vob /vobstore/libpub.vbs
   Locked versioned object base "/vobstore/libpub.vbs".
   ```

6. **Copy the VOB storage directory**—Make sure that the desired parent directory of the target location exists and is writable. Then, copy the entire VOB storage directory tree (but not remote storage pools) to the new location.

   - Same host:

     ```
     <verify that '/src_2/vobstore' already exists>

     # cd /vobstore
     # tar -cf - libpub.vbs | ( cd /src_2/vobstore ; tar -xBpf -)
         (-B option is not necessary (and not supported) on HP-UX systems)
     ```

     (To relocate a VOB storage directory within the same disk partition, you can use a simple mv command.)

- Different host:

  <verify that '/src_2/vobstore' already exists on remote host 'ccsvr04'>

  ```
  # cd /vobstore
  # tar -cf - libpub.vbs | rsh ccsvr04 'cd /src_2/vobstore ;
      tar -xBpf -'
  ```
  (-B option is not necessary (and not supported) on HP-UX systems)

  Note: On some systems, the "remote shell" command has another name (for example, *remsh*). ♦

7. **Ensure that the "old" VOB cannot be reactivated**—Remove it from the ClearCase storage registries:

   ```
   # cleartool rmtag -vob -all /proj/libpub
   # cleartool unregister -vob /vobstore/libpub.vbs
   ```

   This prevents reactivation by ClearCase Release 2 client hosts. If your network also includes client hosts running ClearCase Release 1, prevent them from reactivating the VOB by moving aside the VOB storage directory:

   ```
   # mv /vobstore/libpub.vbs /vobstore/libpub.vbs.OLD
   ```

8. **Terminate the "old" VOB's server processes**—Search the process table for the *vob_server* and *vobrpc_server* processes that manage the "old" VOB. Use *ps -ax* or *ps -ef*, and search for "libpub.vbs". Use *kill*(1) to terminate any such processes.

9. **Register the VOB at its new location**—This example assumes that you are moving a *public* VOB, requiring entry of a password.

   ```
   # cleartool register -vob /net/ccsvr04/src_2/vobstore/libpub.vbs
   # cleartool mktag -vob -public -tag /proj/libpub \
       /net/ccsvr04/src_2/vobstore/libpub.vbs
   Vob tag registry password: <enter password>
   ```

   If your network has several network regions, see "Ensuring the VOB's Global Accessibility" on page 65 for a discussion of adjustments and additional registry entries you may need to make.

10. **Reactivate the VOB**—On all client hosts:

    ```
    % cleartool mount /proj/libpub
    ```

11. **Unlock the VOB.**

```
# rlogin ccsvr04
Password: <enter root password>
# cleartool unlock -vob /src_2/vobstore/libpub.vbs
Unlocked versioned object base
"/src_2/vobstore/libpub.vbs".
```

12. **Delete the old VOB storage directory**—Be sure to first verify that the VOB can be accessed.

```
# rlogin sol
Password: <enter  password>
# rm -fr /vobstore/libpub.vbs          (or 'libpub.vbs.OLD')
```

Moving a VOB (Different Architecture)

This section presents a procedure for moving a *VOB storage directory* to a host with a different architecture. This includes converting the binary-format files that implement the *VOB database*. (To move a VOB to a host of the same architecture, or to another location on the same host, see "Moving a VOB (Same Architecture)" on page 101.) For clarity, we use an example:

- The current location of the VOB storage directory to be moved is */vobstore/libpub.vbs*, on a host named *sol*.
- The VOB is mounted by client hosts at */proj/libpub*.
- The new location for the VOB storage directory is */src_2/vobstore/libpub.vbs* on host *ccsvr04*, whose architecture differs from *sol*'s.

To move the VOB, follow these steps:

1. **Determine whether the VOB has any nonlocal storage pools.**

```
% cleartool lspool -long -vob /proj/libpub | egrep '(^pool|link)'
pool "cdft"
pool "ddft"
pool "sdft"
pool "s_2"
          (remote storage pool)
  pool storage link target pathname
"/net/ccsvr04/ccase_pools/s_2"
      (this pathname must be valid on new VOB host and on all client hosts)
```

2. **Verify the validity of pathnames to nonlocal pools**—Moving a VOB storage directory does not move any of its remote storage pools. You must make sure that the VOB's new host will access each remote storage pool using the same "global pathname" as the VOB's current host:
   - If you are moving the VOB to another location on the same host, this is assured.
   - If you are moving the VOB to a different host, log in to that host and verify that all the remote storage pool global pathnames are valid on that host.
3. **Deactivate the VOB**—On each host where it is currently active, this command deactivates it:

   ```
   % cleartool umount /proj/libpub
   ```
4. **Back up the VOB storage directory**—Use the procedure in "Backing Up a VOB" on page 83 (or in "Backing Up a VOB with Remote Storage Pools" on page 86).
5. **Dump the VOB's database to ASCII dump files**—Do this as the *root* user on the host where the VOB storage directory resides. Don't lock the VOB beforehand—the *reformatvob* command does this automatically).

   ```
   % rlogin sol -l root
   Password: <enter root password>
   # cleartool reformatvob -dump /vobstore/libpub.vbs
     <warning message>
   Reformat versioned object base "/vobstore/libpub.vbs"?
   [no] yes
   Dumping database ...
   .
   .
   Dumped versioned object base "/vobstore/libpub.vbs".
   Done.
   Checking for VOB tag registry entry...
   VOB tag registry entry found for versioned object base
   "/vobstore/libpub.vbs".
   ```

   This command marks the VOB database as invalid. The VOB cannot be used for development work until it is processed by a *reformatvob –load* command.

6. **Copy the VOB storage directory**—First, make sure that the desired parent directory of the target location exists and is writable. Then, copy the entire VOB storage directory tree (but not remote storage pools) to the new location.

   <verify that '/src_2/vobstore' already exists on remote host 'ccsvr04'>

   ```
   # cd /vobstore
   # tar -cf - libpub.vbs | rsh ccsvr04 'cd /src_2/vobstore \
     ; tar -xBpf -'
   ```
   (-B option is not necessary (and not supported) on HP-UX systems)

   Note: On some systems, the "remote shell" command has another name (for example, remsh). ♦

7. **Ensure that the "old" VOB cannot be reactivated**—Remove it from the ClearCase storage registries:

   ```
   # cleartool rmtag -vob -all /proj/libpub
   # cleartool unregister -vob /vobstore/libpub.vbs
   ```

   This prevents reactivation by ClearCase Release 2 client hosts. If your network also includes client hosts running ClearCase Release 1, prevent them from reactivating the VOB by moving aside the VOB storage directory:

   ```
   # mv /vobstore/libpub.vbs /vobstore/libpub.vbs.OLD
   ```

8. **Terminate the "old" VOB's server processes**—Search the process table for the *vob_server* and *vobrpc_server* processes that manage the "old" VOB. Use `ps -ax` or `ps -ef`, and search for "libpub.vbs". Use *kill*(1) to terminate any such processes.

9. **On the new VOB host, recreate the VOB database from the ASCII dump files**—Do this as the *root* user:

   ```
   # rlogin ccsvr04
   Password: <enter root password>
   # cleartool reformatvob -load /src_2/vobstore/libpub.vbs \
     <warning message>
   Reformat versioned object base "/src_2/vobstore/libpub.vbs"?
   [no] yes
   cleartool: Warning: Renamed old database directory to
    "/src_2/vobstore/libpub.vbs/db.01.27".
   cleartool: Warning: Please remove this database backup
    when you are satisfied with the reformat.
   Loading database...
   ```

```
Dumped schema version is nn
 ...
73 pass 2 actions performed.
Done.
Checking for VOB tag registry entry...
cleartool: Warning: VOB tag entry not found for versioned
object base "/src_2/vobstore/libpub.vbs".
cleartool: Warning: Use the mktag command to create a
registry entry.
```

10. **Create a tag for the VOB at its new location**—The *reformatvob* command has already created an entry in the *VOB object* registry; you must create an entry in the *VOB-tag* registry.

    ```
    # cleartool mktag -vob -public -tag \
      /proj/libpub/net/ccsvr04/src_2/vobstore/libpub.vbs
    Vob tag registry password: <enter password>
    ```

    If your network has several network regions, see "Ensuring the VOB's Global Accessibility" on page 65 for a discussion of adjustments and additional registry entries you may need to make.

11. **Reactivate the VOB**—On all client hosts:

    ```
    # cleartool mount /proj/libpub
    ```

12. **Delete the backup VOB database**—This backup was created by *reformatvob –load* in Step #5. It is a datestamped subdirectory of the VOB storage directory. For example, it might be named */src_2/vobstore/libpub.vbs/db.01.27* ("01.27" means "January 27").

    ```
    # rm -fr /src_2/vobstore/libpub.vbs/db.01.27
    ```

13. **Delete the old VOB storage directory**—Be sure to first verify that the VOB can be accessed.

    ```
    # rlogin sol
    Password: <enter  root password>
    # rm -fr /vobstore/libpub.vbs          (or 'libpub.vbs.OLD')
    ```

Removing Unneeded Versions from a VOB

In general, you should approach the removal of source data from VOB storage with extreme caution. Removing entire elements, using *rmelem*, is particularly dangerous:

- Even if an element is no longer needed for the *next* release, you may still need it to reproduce and maintain *previous* releases.
- *rmelem* expunges the element's name from all directory versions in which it was ever cataloged. This "erasing of history" means that the element will not appear in listings or comparisons of old directory versions.
- Making a mistake can be costly—there *is* a procedure for recovering from backup an element that was deleted mistakenly (or in haste), but it's cumbersome. (See "Restoring a Single Element From Backup" on page 110.)

If you need to remove old data in order to conserve disk space, it is far better to remove individual versions of elements, rather than entire elements. The *rmver* command makes it easy to remove versions that you probably won't ever need again. (Another approach is to manipulate storage pools—see "Creating Additional VOB Storage Pools" on page 113.)

By default, *rmver* removes only "uninteresting" versions:

- versions that are unrelated to branching: not located at a branch point, and not the first or last version on a branch
- versions that have no meta-data annotations: version labels, attributes, or hyperlinks

The ClearCase examples directory includes a script that safely removes all "uninteresting" versions of a specified element. This script is located in */usr/atria/examples/rmver_all*. It is particularly useful for deleting automatically-created derived object versions that are no longer needed. (Many organizations check in derived objects regularly, as part of an automated "nightly build" process.)

Example

The following commands illustrate the "before" and "after" of version removal with the *rmver_all* script.

```
% cleartool lsvtree -all ct+register.1
ct+register.1@@/main
ct+register.1@@/main/0
ct+register.1@@/main/1
ct+register.1@@/main/2 (V2.BL2.1, V1.1.4.BL3, V2.BL2)
ct+register.1@@/main/3 (RGYWORK_BASE)
ct+register.1@@/main/rgywork
ct+register.1@@/main/rgywork/0
ct+register.1@@/main/rgywork/1
ct+register.1@@/main/rgywork/2
ct+register.1@@/main/rgywork/3
ct+register.1@@/main/4
ct+register.1@@/main/5 (V2.BL3)
ct+register.1@@/main/6
ct+register.1@@/main/7
ct+register.1@@/main/8
ct+register.1@@/main/9 (V2.BL4_PRINT, V2.BL4)

% /usr/atria/examples/rmver_all/rmver_all ct+register.1
Removed these versions of "ct+register.1":
  /main/rgywork/1
  /main/rgywork/2
  /main/1
  /main/4
  /main/6
  /main/7
  /main/8

% cleartool lsvtree -all ct+register.1
ct+register.1@@/main
ct+register.1@@/main/0
ct+register.1@@/main/2 (V2.BL2.1, V1.1.4.BL3, V2.BL2)
ct+register.1@@/main/3 (RGYWORK_BASE)
ct+register.1@@/main/rgywork
ct+register.1@@/main/rgywork/0
ct+register.1@@/main/rgywork/3
ct+register.1@@/main/5 (V2.BL3)
ct+register.1@@/main/9 (V2.BL4_PRINT, V2.BL4)
```

Restoring a Single Element From Backup

If you mistakenly delete an element with *rmelem*, you can restore it from a backup tape, using the procedure presented in this section.

Note: Mistakenly removing a directory element does *not* remove the file elements cataloged within it—file elements exist independently of directory elements. In many cases, deleting a directory element causes the files within it to be transferred to the VOB's *lost+found* directory. ♦

The overall procedure for restoring an element is:

- Unmount the VOB whose element has been deleted.
- Restore the "old" VOB storage directory from the backup medium to a temporary location on disk.
- Mount the "old" VOB.
- Create a new temporary VOB and mount it.
- Use *clearcvt_ccase* to "copy" the element from the old VOB to the temporary VOB.
- Unmount the old VOB.
- Remount the real VOB.
- Use *clearcvt_ccase* to "copy" the element from the temporary VOB to the real VOB.
- Delete the "old" VOB and the temporary VOB.

As an example, suppose that file element *util.c* has inadvertently been deleted from directory */vobs/proj/src*. The VOB-tag is */vobs/proj*, and the VOB storage directory is */vobstore/proj.vbs* on the local host. Here's how a VOB's owner can restore the element from a backup tape.

1. **Unmount the VOB whose element has been deleted**—This is essential, because two copies of the same VOB must never be active at the same time.

   ```
   % cleartool umount /vobs/proj
   ```

2. **Remove the VOB's storage registry entries**—These entries would prevent use of an old version of the same VOB.

```
% cleartool rmtag -vob /vobs/proj
% cleartool unregister -vob /vobstore/proj.vbs
```

3. **Terminate the VOB's server processes**—Search the process table for the ClearCase *vob_server* and *vobrpc_server* processes that manage that VOB. Use *ps -ax* or *ps -ef*, and search for "/vobstore/proj.vbs"; use *kill*(1) to terminate any such processes. (Only the *root* user can kill a *vobrpc_server* process.)

4. **Restore the "old" VOB storage directory**—Suppose that the backup tape was created with *tar*. Restore the "old" VOB to a temporary location, not to its original location.

```
% cd /usr/tmp
% tar -xf /dev/tape
```

   This restores the VOB storage directory tree as */usr/tmp/proj.vbs*.

5. **Register and mount the "old" VOB**—As in the preceding step, use a temporary mount point, not the original mount point.

```
% cleartool register -vob /usr/tmp/proj.vbs
% cleartool mktag -vob -tag /tmp/oldvob /usr/tmp/proj.vbs
% mkdir /tmp/oldvob
% cleartool mount /tmp/oldvob
```

6. **Create a new temporary VOB and mount it**—Be sure *not* to give this VOB a *public* VOB-tag.

```
% cleartool mkvob -nc -tag /tmp/newtmpvob /usr/tmp/newtmpvob.vbs
Created versioned object base.
  .
  .
% mkdir /tmp/newtmpvob
% cleartool mount /tmp/newtmpvob
```

7. **"Copy" the element: old VOB to temporary VOB—**Use the *clearcvt_ccase* program to make a complete copy of the element.

```
    (Create conversion script in "old" VOB)
% cleartool setview dfl
    (set a view configured with the default config spec)
% cd /tmp/oldvob/src
% clearcvt_ccase util.c
Converting element "util.c" ...
```

```
Extracting element history ...
  .
  .
Creating script file cvt_dir/cvt_script ...
     (Create conversion script in temporary VOB)
% cd /tmp/newtmpvob
% /tmp/oldvob/src/cvt_dir/cvt_script
Converting files from /tmp/oldvob/src to .
  .
  .
Checked in "./util.c" version "/main/3".
Checked in "./." version "/main/2".
```

8. **Unmount and unregister the old VOB**—This corresponds to the work done in Steps #1 and #2.

```
% cd /
% cleartool umount /tmp/oldvob
% cleartool rmtag -vob /tmp/oldvob
% cleartool unregister -vob /usr/tmp/proj.vbs
```

9. **Terminate the old VOB's server processes**—This is similar to Step #3. This time, search the process table for a *vob_server* and/or *vobrpc_server* invoked with "/usr/tmp/proj.vbs".

10. **Reregister and remount the real VOB**—This time, make a public VOB-tag.

```
% cleartool register -vob /vobstore/proj.vbs
% cleartool mktag -vob -public -tag /vobs/proj /vobstore/proj.vbs
Vob tag registry password: <enter password>
% cleartool mount /vobs/proj
```

11. **"Copy" the element: temporary VOB to original VOB**—Use the *clearcvt_ccase* program to make a complete copy of the element.

```
     (Create conversion script in "old" VOB)
% cd /tmp/newtmpvob
% clearcvt_ccase util.c
Converting element "util.c" ...
Extracting element history ...
  .
  .
Creating script file cvt_dir/cvt_script ...
     (Create conversion script in temporary VOB)
% cd /vobs/proj/src
% /tmp/newtmpvob/cvt_dir/cvt_script
```

```
Converting files from /tmp/newtmpvob to .
  .
  .
Checked in "./util.c" version "/main/3".
Checked in "./." version "/main/5".
```

12. **Unmount the temporary VOB**—This corresponds to the work done in Step #6.

    ```
    % cd /
    % cleartool umount /tmp/newtmpvob
    ```

13. **Delete the temporary VOB**—Use *rmvob*, which automatically removes the VOB's registry entries and terminates all of its server processes.

    ```
    % cleartool rmvob /usr/tmp/newtmpvob.vbs
    Remove versioned object base "/usr/tmp/newtmpvob.vbs"
    [no] yes
    Removed versioned object base "/usr/tmp/newtmpvob.vbs".
    ```

Creating Additional VOB Storage Pools

If a VOB is threatening to fill up its disk partition, you may want to rework the VOB's storage pools. The VOB database (*db* subdirectory) must be physically located within the VOB storage directory, but any or all of its storage pools can be remote. In the extreme case, you might "abandon" the VOB's default storage pools, and transfer all of the VOB's file system data to other disk partitions and/or remote hosts.

The host on which you create a new storage pool need not have ClearCase installed. It can have any hardware/software architecture, but the pool must be NFS-accessible at the same pathname from all ClearCase hosts (for example, */net/ccsvr02/ccase_pools/do_3*). See "Default, Local, and Remote Storage Pools" on page 15 for additional information.

Note: The ClearCase *network region* facility does not apply here. The pathname of a remote storage pool must be truly "global". ♦

When deciding which host(s) to use for new storage pools, consider that usage patterns vary with the kind of storage pool:

- **Source pools**—These pools store the most precious data: the checked-in versions of file elements. Traffic involving these pools is relatively light,

but data integrity is very important. (See "Caution on Remote Source Pools" below.) The ideal location for such pools is a robust file server, with a large capacity and frequent, reliable data backups.

- **Cleartext pools**—These pools will probably get the heaviest traffic (assuming that many of your file elements are stored in *delta* and/or compressed format). But the data in cleartext pools is completely expendable, since ClearCase can reconstruct it at any time. The ideal location for such pools is a machine with a fast file system.
- **Derived object pools**—These pools can grow to be quite large, depending on the number of active configurations (three new development projects, two old releases in maintenance, and so on). Anticipate the storage requirements in the new pool for each active configuration; make sure the disk partition can handle the total storage requirement.

The principal tools for working with storage pools are:

lspool	List a VOB's existing storage pools.
describe	List an element's storage pool assignments. You must use the extended naming symbol (@@) when specifying the element: `% cleartool describe getcwd.c` *(wrong)* `% cleartool describe getcwd.c@@` *(right)*
mkpool	Create a new storage pool. Use the -ln option to create a remote storage pool.
chpool	Reassign a file or directory element to another storage pool. (A directory element itself is stored entirely within the VOB database; a directory's pool assignments control which pools will be used by new file elements and derived objects created within the directory.)

Caution on Remote Source Pools

We recommend that you keep *source* storage pools local, within the VOB storage directory. This strategy optimizes data integrity—a single disk partition will contain all of the VOB's essential data. It will also simplify backup/restore procedures. This concern typically overrides performance considerations, since losing a source pool means that developers must recreate the lost versions.

Example: Assigning All Files in a Directory to a New Pool

The following example shows how to create a new, remote source storage pool, and then reassign all the current and future elements in a particular directory to the new pool.

1. **Create the new storage pool**—Be careful to specify a global pathname for the remote pool.

```
% cd /vobs/bgr
% cleartool mkpool -source -ln \
  /net/ccsvr02/ccase_pools/bgrsrc2 bgrsrc2
Comments for "bgrsrc2":
remote source storage pool
.
Created pool "bgrsrc2".
```

2. **Reassign existing file elements to the new pool**—In this example, we reassign all the file elements in a particular development subdirectory.

```
% cd libbgr
% cleartool find . -type f -exec 'cleartool chpool -force\
  bgrsrc2 $CLEARCASE_PN'
Changed pool for "./Makefile" to "bgrsrc2".
Changed pool for "./errmsg.c" to "bgrsrc2".
Changed pool for "./fork3.c" to "bgrsrc2".
Changed pool for "./get.c" to "bgrsrc2".
Changed pool for "./getcwd.c" to "bgrsrc2".
  .
  .
Changed pool for "./stint.h" to "bgrsrc2".
Changed pool for "./strut.c" to "bgrsrc2".
```

3. **Reassign the directory element to the new pool, too**—This will cause all newly-created file (and directory) elements in this directory to use the new pool, also.

```
% cleartool chpool bgrsrc2 .
Changed pool for "." to "bgrsrc2".
```

Example: Moving an Existing Storage Pool to Another Disk

There is no built-in command for moving an existing storage pool. ClearCase routines perform standard UNIX operations to access storage pools and the data containers within them. Thus, you can move a storage pool in this way:

1. **Determine the location of the storage pool**—Use the *lspool* command:

   ```
   % cleartool lspool -long -vob /vobs/bgr d_aux \
     pool "d_aux"
    .
    .
     pool storage global pathname
         "/net/ccsvr01/vobstore/bgr.vbs/d/d_aux"
   ```

2. **Lock the VOB**—This will prevent any new shared DOs from being placed in the storage pool while you are working on it:

   ```
   % cleartool lock -vob /vobs/bgr
   Locked versioned object base
   "/net/ccsvr01/vobstore/bgr.vbs".
   ```

3. **Copy the contents of the storage pool**—The storage pool is a standard UNIX directory. You can copy its contents to a new location using *cp*, *rcp*, *tar*, or other commands. For example:

   ```
   % rlogin ccsvr01
   % mkdir -p /vobstore_2/DO_pools
   % cp -r /vobstore/bgr.vbs/d/d_aux /vobstore_2/DO_pools
   ```

4. **Replace the old storage pool with a symbolic link**—Move the old storage pool aside, then create the link in its place.

   ```
   % cd /vobstore/bgr.vbs/d
   % mv d_aux d_aux.MOVED
   % ln -s /net/ccsvr01/vobstore_2/DO_pools/d_aux d_aux
   ```

   Be sure the text of the symbolic link is a globally-valid pathname to the new storage pool location.

5. **Unlock the VOB.**

   ```
   % cleartool unlock -vob /vobs/bgr
   Unlocked versioned object base
   "/net/ccsvr01/vobstore/bgr.vbs".
   ```

6. **Remove the old storage pool**—When you have verified that the storage pool is working well in its new location, you can remove the old pool:

   ```
   % rm -fr /vobstore/bgr.vbs/d/d_aux.MOVED
   ```

Adjusting Storage Pool Scrubbing

Chapter 10 discusses the standard maintenance procedures established for each host when ClearCase is installed there. There are two typical motivations for adjusting a VOB host's default procedures for storage pool scrubbing:

- **Not enough space**—The disk partitions in which VOB storage pools reside may be chronically filling up. This would indicate adoption of a more aggressive scrubbing strategy.
- **Not enough time**—The *scrubber* utility may be taking too much time to complete, interfering with other overnight activities, such as nightly software builds. This would indicate adoption of a less aggressive scrubbing strategy.

You may find it necessary to do some experimentation. For example, adjusting scrubbing to take place less frequently may cause disk-space problems that you had not previously experienced. Before making any scrubbing adjustments on a VOB host, be sure to analyze its */usr/adm/atria/log/scrubber_log* file. The *scrubber* manual page explains how to read this file.

The following sections present some simple examples of adjusting the way VOB storage pools are scrubbed.

Scrubbing Derived Objects More Often

By default, *scrubber* allows data containers of zero-referenced derived objects to remain in their storage pools for 4 days (96 hours). If DOs are filling up a VOB's disk partition, you might shorten this "grace period". Here's a command that causes a VOB's default DO storage pool (*ddft*) to be emptied of unneeded data containers every day (that is, every 24 hours):

```
% cleartool mkpool -update -vob VOB-tag -age 24 ddft
Updated pool "ddft".
```

Fine-Tuning Derived Object Scrubbing

Suppose that the adjustment in the preceding section is not enough to keep the disk partition from filling up. You might decide to invoke the *scrubber* utility more often: during the work day as well as overnight. To minimize the impact on users during the work day, you might pinpoint the scrubbing—perhaps to the DOs created in a particular directory, */vobs/proj/reorg*:

1. **Determine the directory's current DO storage pool assignment**—You will need to clean up this storage pool.

   ```
   % cd /vobs/proj
   % cleartool describe -long reorg@@
   directory element "reorg@@":
     .
     .
     ...  derived pool: ddft
       (this directory uses the VOB's default DO storage pool)
   ```

2. **Assign the directory to a separate storage pool**—This will enable pinpoint control of scrubbing, which can be invoked on a per-pool basis:

   ```
   % cd /vobs/proj
   % cleartool mkpool -derived new_do_pool
   Comments for "new_do_pool":
   pool for DOs created in /vobs/proj/reorg
   .
   Created pool "new_do_pool".
   % cleartool chpool new_do_pool reorg
   Changed pool for "reorg" to "new_do_pool".
   ```

3. **Determine the location of the VOB storage directory**—You'll need the pathname of the VOB's storage directory for *scrubber*. Use *lsvob* to determine the pathname:

   ```
   % cleartool lsvob /vobs/proj
   * /vobs/proj     /net/ccsvr03/vobstore/proj.vbs
   ```

4. **Have the new storage pool scrubbed thoroughly and often**—There are many ways to accomplish this. You might take advantage of the fact the default maintenance procedure invokes a script named */usr/atria/config/cron/ccase_local.day* once a day (as part of the 4:30 am regimen). You can create or modify this script to set up several *at*(1) jobs:

```
% su
Password: <enter root password>
# vi /usr/atria/config/cron/ccase_local.day

 <add these lines:>

CMD="/usr/atria/etc/scrubber -e -p new_do_pool \
    /net/ccsvr03/vobstore/proj.vbs"
echo $CMD | at 10:00
echo $CMD | at 13:00
echo $CMD | at 15:00
echo $CMD | at 18:00

 <save the file and end the edit session>

# exit
%
```

The *scrubber* utility will be invoked four times during the work day on the derived object storage pool *new_do_pool*. The *-e* option to *scrubber* empties the pool of all zero-referenced DOs.

5. **Clean up the old DO storage pool**—The *chpool* command in Step #2 does not move existing DO data containers; it only affects where a new DO's data container will be stored. Accordingly, you should clean up the old storage pool:

   ```
   % /usr/atria/etc/scrubber -e -p ddft \
     /net/ccsvr03/vobstore/proj.vbs
   ```

Caution: Depending on how ClearCase is installed, the directory */usr/atria/config/cron* may or may not actually be local to a particular host. If two or more hosts share the same */usr/atria/config/cron* directory, you may need to have the "local" script perform conditional processing, based on the hostname. Alternatively, you can use a completely separate mechanism to invoke supplementary maintenance procedures.

Scrubbing Less Aggressively

If the 4:30 am scrubbing regimen takes too long (perhaps spilling over into the work day). you might make the starting time earlier. Alternatively, you might change the way that scrubber is invoked, so that it takes less time to run. Both these alternatives involve a change to */usr/atria/config/cron*; hence, the "Caution" paragraph above applies in this situation, too. Here's how you might revise scrubbing to process DO pools only, leaving cleartext pools alone:

```
% su
Password: <enter root password>
# vi /usr/atria/config/cron/
```

revise this line ...

```
$ATRIA_DIR/etc/scrubber -f -a
```

... to read:

```
$ATRIA_DIR/etc/scrubber -f -a -k do
```

Chapter 12

Occasional View Maintenance

This chapter presents step-by-step procedures for a variety of view-maintenance tasks.

Moving a View (Same Architecture)

This section presents a procedure for moving a *view storage directory* to another location, either on the same host or on another host with the same architecture. (To move a view to a host of a different architecture, see "Moving a View (Different Architecture)" on page 123.) For clarity, we use an example:

- The current location of the view storage directory to be moved is */users/gomez/viewstore/gomez.vws*, on a host named *earth*.
- The new location for the view storage directory is */public/gomez.vws*. We consider two cases: (1) the new location is also on *earth*; (2) the new location is on another host, named *ccsvr04*.

To move the view, follow these steps:

1. **Go to the view's host**—Log in as the view's owner:

   ```
   % rlogin earth -l gomez
   ```

2. **Determine whether the view has a nonlocal private storage area.**

   ```
   % ls -ld /users/gomez/viewstore/gomez.vws/.s /
       ...   .s -> /public/view_aux/gomez
   ```

   (a symbolic link indicates that the private storage area is remote)

3. **Deactivate the view**—Use the *ps*(1) command to determine the process-ID of the view's *view_server* process. Then, use the *kill*(1) command to terminate that process.

4. **(if necessary) Validate the private storage area's global pathname**—This step is required only if the view's private storage area is remote, *and* you are moving the view to another host. You must verify that the view's new host can access the private storage area using the same "global pathname" as the view's current host:

   ```
   % rlogin ccsvr04
   % ls /public/view_aux/gomez
     .
     .  (this command should succeed)
     .
   % exit
   ```

   If the intended destination host cannot access the view's private storage area in this way, select and validate another host.

5. **Back up the view storage directory**—Use the procedure in "Backing Up a View" on page 91.

6. **Copy the view storage directory**—First, make sure that the desired parent directory of the target location exists and is writable. Then, copy the entire view storage directory tree (but not a remote private storage area) to the new location.

   - Same host:

     <verify that '/public' already exists>

     ```
     % cd /users/gomez/viewstore
     % tar -cf - gomez.vws | ( cd /public ; tar -xBpf - )
         (-B option is not necessary (and not supported on HP-UX systems)
     ```

   - **Different host:**

     <verify that '/public' already exists on remote host 'ccsvr04'>

     ```
     % cd /users/gomez/viewstore
     % tar -cf - gomez.vws | rsh ccsvr04 'cd /public; \
         tar -xBpf -'
         (-B option is not necessary (and not supported on HP-UX systems)
     ```

     Note: On some systems, the "remote shell" command has another name (for example, *remsh*). ♦

7. **Ensure that the "old" view cannot be reactivated**—Remove it from the ClearCase storage registries:

   ```
   # cleartool rmtag -view -all gomez
   # cleartool unregister -view /users/gomez/viewstore/gomez.vws
   ```

   This prevents reactivation by ClearCase Release 2 client hosts. If your network also includes client hosts running ClearCase Release 1, prevent them from reactivating the view by moving aside the view storage directory:

   ```
   % cd /users/gomez/viewstore
   % mv gomez.vws gomez.vws.OLD
   ```

8. **Register the view at its new location.**

   ```
   % cleartool register -view /public/gomez.vws
   % cleartool mktag -view -tag gomez /public/gomez.vws
   ```

   If your network has several network regions, you need to make additional registry entries. This procedure is essentially similar to the one in "Ensuring the VOB's Global Accessibility" on page 65.

9. **Reactivate the view.**

   ```
   % cleartool startview gomez
   ```

10. **Delete the old view storage directory**—Be sure to first verify that the view can be accessed at its new location.

    ```
    % rm -fr /users/gomez/viewstore/gomez.vws
    ```

    Note: Moving a view does not modify the *.view* file in the view storage directory. The information in this file always describes the view "first incarnation". ♦

Moving a View (Different Architecture)

This section presents a procedure for moving a *view storage directory* to a host with a different architecture. This includes converting the binary-format files that implement the *view database*. (To move a view to a host of the same architecture, or to another location on the same host, see "Moving a View (Same Architecture)" on page 121.) For clarity, we use an example:

- The current location of the view storage directory to be moved is */users/gomez/viewstore/gomez.vws*, on a host named *earth*.

- The new location for the view storage directory is */public/gomez.vws* on host *ccsvr04*, whose architecture differs from *earth*'s.

To move the view, follow these steps:

1. **Go to the view's host**—Log in as the view's owner:

   ```
   % rlogin earth -l gomez
   ```

2. **Determine whether the view has a nonlocal private storage area.**

   ```
   % ls -ld /users/gomez/viewstore/gomez.vws/.s
      ...   .s -> /public/view_aux/gomez
   ```
   (a symbolic link indicates that the private storage area is remote)

3. **Deactivate the view**—Use the *ps*(1) command to determine the process-ID of the view's view_server process. Then, use the *kill*(1) command to terminate that process.

4. **(if necessary) Validate the private storage area's global pathname**—This step is required only if the view's private storage area is remote, *and* you are moving the view to another host. You must verify that the view's new host can access the private storage area using the same "global pathname" as the view's current host:

   ```
   % rlogin ccsvr04
   % ls /public/view_aux/gomez
     .
     .  (this command should succeed)
     .
   % exit
   ```

   If the intended destination host cannot access the view's private storage area in this way, select and validate another host.

5. **Back up the view storage directory**—Use the procedure in "Backing Up a View" on page 91.

6. **Dump the view's database to ASCII dump files.**

   ```
   % cleartool reformatview -dump /users/gomez/viewstore/gomez.vws
   ```

   This creates files *view_db.dump_file* and *view_db.state* in the view storage directory. It also renames the view database subdirectory to *db.dumped*.

7. **Copy the view storage directory**—First, make sure that the desired parent directory of the target location exists and is writable. Then, copy the entire view storage directory tree (but not a remote private storage area) to the new location.

   Note: On some systems, the "remote shell" command has another name (for example, *remsh*). ♦

   <verify that '/public' already exists on remote host 'ccsvr04'>

   ```
   % cd /users/gomez/viewstore
   % tar -cf - gomez.vws | rsh ccsvr04 'cd /public; tar -xBpf -'
   ```
   (-B option is not necessary (and not supported on HP-UX systems)

8. **Ensure that the "old" view cannot be reactivated**—Remove it from the ClearCase storage registries:

   ```
   # cleartool rmtag -view -all gomez
   # cleartool unregister -view /users/gomez/viewstore/gomez.vws
   ```

   This prevents reactivation by ClearCase Release 2 client hosts. If your network also includes client hosts running ClearCase Release 1, prevent them from reactivating the view by moving aside the view storage directory:

   ```
   % cd /users/gomez/viewstore
   % mv gomez.vws gomez.vws.OLD
   ```

9. **On the new host, recreate the view database from the ASCII dump files.**

   ```
   % rlogin ccsvr04
   % cleartool reformatview -load /public/gomez.vws
   ```

10. **Register the view at its new location.**

    ```
    % cleartool register -view /public/gomez.vws
    % cleartool mktag -view -tag gomez /public/gomez.vws
    ```

    If your network has several network regions, you need to make additional registry entries. This procedure is essentially similar to the one in "Ensuring the VOB's Global Accessibility" on page 65.

11. Reactivate the view.

    ```
    % cleartool startview gomez
    ```

12. **Delete the backup view database**—This backup, named *db.dumped*, was created by *reformatview –load* in Step #6.

    ```
    % rm -fr /public/gomez.vws/db.dumped
    ```

13. **Delete the old view storage directory**—Be sure to first verify that the view can be accessed at its new location.

    ```
    % rm -fr /net/sol/users/gomez/viewstore/gomez.vws
    ```

Note: Moving a view does not modify the *.view* file in the view storage directory. The information in this file always describes the view "first incarnation". ♦

Moving a View's Private Storage Area

Use the following procedure to move a view's private storage area to another location. For clarity, we use an example involving view storage directory */users/gomez/viewstore/gomez.vws*, on host *earth*. The procedure works both when the private storage area is local (*.s* is an actual subdirectory of the view storage directory) and when it is remote (*.s* is a UNIX-level symbolic link to a remote location).

1. **Go to the view's host**—Log in as the view's owner:

   ```
   % rlogin earth -l gomez
   ```

2. **Deactivate the view**—Use the *ps*(1) command to determine the process-ID of the view's *view_server* process. Then, use the *kill*(1) command to terminate that process.

3. **Go to the private storage area**—The private storage area is standard UNIX directory tree, with *.s* as its root.

   ```
   % cd /users/gomez/viewstore/gomez.vws/.s
   ```

4. **Copy the entire directory tree**—You can copy the directory tree to a new location using *cp*, *rcp*, *tar*, or other commands. For example:

   ```
   % mkdir -p /public/view_aux/gomez.priv
   % cp -r * /public/view_aux/gomez.priv
   ```

   Be sure to select a new location that is globally accessible.

5. **Replace the old .s directory with a symbolic link**—It doesn't matter whether the existing *.s* is an actual subdirectory or a symbolic link. Just move it aside and create a (new) symbolic link in its place:

   ```
   % cd ..
   % mv .s .s.MOVED
   % ln -s /public/view_aux/gomez.priv .s
   ```

6. **Reactivate the view**—Use *startview* or *setview.*

7. **Remove the private storage area**—When you have verified that the private storage pool is working well in its new location, you can remove the old one:

   - If old private storage area *.s.MOVED* is an actual directory:

     ```
     % rm -fr /users/gomez/viewstore/gomez.vws/.s.MOVED
     ```

   - If old private storage area *.s.MOVED* is a UNIX symbolic link:

     ```
     % cd /users/gomez/viewstore/gomez.vws/.s.MOVED
     % rm -fr *
     % cd ..
     % rmdir .s.MOVED
     ```

Manual Cleanup of a View

Use the following procedure to remove unwanted files from a view's private storage area. (You may wish to adapt this procedure to your own organization's needs, and then publicize it to all ClearCase users.)

Suppose the view's view-tag is R2integ.

1. **Set the view.**

   ```
   % cleartool setview R2integ
   ```

2. **Take inventory of the view's private files with** *lsprivate*—The *lsprivate* command lists view-private files using the pathnames at which they appear in VOBs:

   ```
   % cleartool lsprivate | tee /tmp/R2integ.lsp
   /vobs/proj/sun4/pick.o
   /vobs/proj/sun4/spar.o
   /vobs/proj/lib/get.c  [checkedout]
   /vobs/proj/lib/get.c~
   ```

```
/vobs/proj/lib/querytty.c  [checkedout]
/vobs/proj/lib/querytty.c~
/vobs/proj/lib/strut.c  [checkedout]
  .
  .
```

Be sure to place the output in a scratch "inventory" file, as in this example. Don't worry if some `not available - deleted perhaps?` error messages appear. Such messages are sent to *stderr*; corresponding information is also written to *stdout*, and thus will be captured in the scratch file.

3. **Extract the names of unneeded files**—Use a text editor, the *grep* utility, or any other tool to extract from the scratch file the names of files that can safely be deleted. Write this list to another file—for example, */tmp/R2integ.deleteme*. Likely candidates include:

   - text-editor temporary files and backup files (*foo.c~*, *#foo.c#*, *foo.c.backup*, and so on)
   - output files produced by text formatters (*foo.ps*, *foo.dvi*, and so on)
   - core dump files

   Be sure to *exclude* from the to-be-deleted list any checked-out files. Such files are annotated with `[checkedout]` in the *lsprivate* output, as shown above.

4. **Double-check the list**—Make sure it contains only files to be deleted.

5. **Delete the view-private files**—Use the shell's "command substitution" feature to delete the files in the extracted list:

   ```
   % rm 'cat /tmp/R2integ.deleteme'
   ```

   Note: The following steps are appropriate only if `not available - deleted perhaps?` error messages appeared in Step #2 above. ♦

6. **Decide which stranded files should be deleted**—The error messages, and corresponding lines with *VOB–* and/or *DIR–* in the inventory file, describe *stranded* view-private files. Such files belong to VOBs and/or directories that are not currently accessible—and, in some cases, may never become accessible again. Consult the *lsprivate* manual page to learn more about stranded files, and to decide which files should be deleted. In general, you don't select *individual* files, but entire directories or entire VOBs, all of whose view-privates files are to be deleted.

7. **Collect the appropriate UUIDs**—Determine the UUID of each VOB directory and each VOB whose files are to be deleted. For example, the following lines from *lsprivate* output describes a stranded file named *hello.c.ann*:

   ```
   <VOB-beeb313c.0e8e11cd.ad8e.08:00:69:06:af:65>/
       <DIR-375b5ca0.0e9511cd.ae20.08:00:69:06:af:65>/hello.c.ann
   ```

   (Here, we've split the line for readability—it is not split in *lsprivate*'s output.) The VOB from which the file is stranded has UUID `beeb313c.0e8e11cd.ad8e.08:00:69:06:af:65`; the VOB directory in which the stranded file was created has UUID `375b5ca0.0e9511cd.ae20.08:00:69:06:af:65`.

8. **Move stranded files to the view's *lost+found* directory**—To remove a set of stranded files, first transfer them to the view's special *lost+found* directory, using the *recoverview* command. For example, this command transfers all stranded view-private files created in the same directory as *hello.c.ann:*

   ```
   % cleartool recoverview -dir
   375b5ca0.0e9511cd.ae20.08:00:69:06:af:65 -tag R2integ
   Moved file
   /net/ccsvr03/vus/integ/.s/lost+found/57FBB6DF.0418.util.c.ann
   Moved file
   /net/ccsvr03/vus/integ/.s/lost+found/2203B56D.00C2.hello.c.ann
   ```

   In this example, *recoverview* transfers just two files, *util.c.ann* and *hello.c.ann*, to the *lost+found* directory.

9. **Delete the files from the *lost+found* directory**—You can now use a standard *rm*(1) command to delete the stranded files:

   ```
   % cd /net/ccsvr03/vus/integ/.s/lost+found
   % rm 57FBB6DF.0418.util.c.ann
   % rm 2203B56D.00C2.hello.c.ann
   ```

Chapter 13

ClearCase Performance Tuning

This chapter presents techniques for improving ClearCase performance. There are techniques for addressing performance issues at the host level, at the VOB level, and at the view level.

Improving VOB Host Performance

Your organization's VOBs constitute a central data repository. Good VOB host performance ensures that the centralized resource does not become a bottleneck.

Although a VOB appears to be a version-smart file server, its implementation involves significant database access and computation. VOB usage patterns can greatly influence how many concurrent users will experience good ClearCase performance. For example, many more users can read header files from a VOB directory at a level of good performance than can produce derived objects in a similar directory.

Eliminate Extraneous Processes

The most effective measures for ensuring good performance from VOB hosts are also the easiest to implement (technically, if not organizationally):

- **Keep non-ClearCase processes off the VOB host**—Don't have the VOB host also serve as a server host for another application (for example, a DBMS), or at the system-level (for example, as an NIS server).
- **Keep ClearCase client processes off the VOB host**—Make sure that no one is performing *clearmake* builds on any VOB host. Similarly, make sure no one is using other client tools: *cleartool*, *xclearcase*, *xcleardiff*, and so on.

- **Keep** *view_server* **processes off the VOB host**—This recommendation may be harder to implement; many organizations create shared views on the same hosts as VOBs. If possible, minimize this double-usage of VOB hosts.

  Exception: For reliable *non-ClearCase access* (avoiding "multihop" network access paths), place the VOB and the view through which it is exported on the *same* host. For more information, see "Setting Up an Export View for Non-ClearCase Access" on page 74 and the *exports_ccase* manual page.

Manipulate Block Buffer Caches

All the UNIX-based operating systems supported by ClearCase have a *dynamic block buffer cache* feature. As much main memory as possible is used to cache blocks of data files that have been updated by user processes. Periodically, the contents of the block buffer cache is *flushed* to disk.

This feature speeds up disk I/O significantly; making full use of it is a very important factor in good VOB host performance. An inadequate block buffer cache causes thrashing of VOB database files—the files in the *db* subdirectories of VOB storage directories). The result is a significant performance degradation, evidenced by:

- extended periods required for *scrubber* and *vob_scrubber* execution
- very slow *clearmake* builds
- ClearCase clients getting RPC timeouts

We recommend that the size of a VOB host's block buffer cache average about 200% of the size of the host's largest VOB database file; the minimum acceptable size is about 50%. You cannot directly control the size of the block buffer cache; its size increases automatically when you add more main memory to the host.

If there is a substantial amount on non-ClearCase activity and/or ClearCase client activity on the host, you will need even more main memory to assure good VOB database performance.

Block Buffer Cache Statistics

The standard UNIX System V *sar*(1M) utility reports block buffer cache activity. For example, this command reports activity over a 5-minute period, with a cumulative sample taken every 60 seconds:

```
% sar -b 60 5
12:14:22 bread/s lread/s  %rcache bwrit/s lwrit/s  %wcache pread/s pwrit/s
12:15:22       0       1      100       1       1        0       0       0
12:16:23       1       1      -60       2       2        0       0       0
12:17:24       0       4      100       4      17       77       0       0
12:18:25       0       6      100       3     145       98       0       0
12:19:25      17      91       81      28     335       92       0       0

12:19:25 bread/s lread/s  %rcache bwrit/s lwrit/s  %wcache pread/s pwrit/s
Average        4      21       83       8     100       92       0       0
    (cache-reads should be in the 90%–95% range)
    (cache-writes should be 75% or above)
```

Some UNIX variants provide special tools for monitoring buffer cache performance. For example, IRIX has *osview*; HP-UX has *glance*.

Flushing of the Block Buffer Cache

Interactive performance suffers considerably when the block buffer cache is flushed to disk. Most UNIX variants provide no user-level control over the frequency of flushing; HP-UX does, through the *syncer*(1M) utility. The larger the block buffer cache, the less frequently it should be flushed.

Improving Client Host Performance

Performance of a ClearCase client host can be adjusted at the client program level, at the *view_server*, and/or at the MVFS level.

Increasing System Resources

Client workstations supporting a single user should have a minimum of 10–15 MIPS processing power, 16Mb of main memory, and 300Mb of disk storage. An additional 8–16Mb of main memory will further improve performance. Extra memory is especially recommended for users who run memory-intensive applications in the ClearCase environment, make

extensive use of graphical user interfaces, or want their client workstations to serve double-duty as hosts for parallel distributed building.

Creating Remote Storage Pools

The ClearCase default is to store all of a VOB's file system data in the default storage pools created by *mkvob*. These pools are located within the VOB storage directory. If a VOB host become I/O-bound, it is probably due to high storage pool traffic, caused by either "too many users" or "too many files".

You can supplement (or replace) the default pools with *remote storage pools*, which effectively enable a VOB to outgrow its storage directory's disk partition. Remote pools need not be located on ClearCase hosts; they need only be accessible through NFS.

In some situations, remote storage pools can improve performance, as well:

- If a particular view is being used heavily (perhaps by a group performing integration work), build performance may improve if the cleartext and derived object storage pools involved in the builds are located on the same host as the view storage directory.
- Faster access to any storage pool may be achieved if it is located on a server host with a very fast file system.

Caution on Remote Source Pools

We recommend that you keep source pools local, within the VOB storage directory. This strategy optimizes data integrity—a single disk partition will contain all of the VOB's essential data. It will also simplify backup/restore procedures. This concern typically overrides performance considerations, since losing a source pool means that developers must recreate the lost versions.

If source pool access produces a significant processor or I/O bottleneck, you might temporarily move some elements into source pools on different hosts.

See "Creating Additional VOB Storage Pools" on page 113 for a step-by-step procedure.

Changing the MVFS Configuration (SunOS Only)

This section describes procedures for reconfiguring the ClearCase *multiversion file system* (MVFS) on hosts running SunOS 4 or SunOS 5 (Solaris). By default, the MVFS is dynamically loaded at system startup with the following configuration:

- MVFS-internal identifiers (*mnodes*) cached for up to 4096 MVFS objects
- up to 900 unused *mnode* numbers cached
- UNIX-internal identifiers (*vnodes*) cached for up to 100-400 cleartext files, depending on the system-wide maximum number of users (MAXUSERS kernel configuration parameter)
- up to 1400 names of MVFS objects cached

You may wish to change the MVFS cache sizes to improve performance if your host performs builds that involve a large number of files, as indicated in Table 13-1.

Table 13-1 Selecting the Default or Alternative MVFS Cache Configuration

Main Memory	Files Used in Typical Build	Recommended MAXUSERS Value	Recommended MVFS Cache Configuration
16Mb	any	16	default
24Mb	any	32	default
32Mb	< 400	32	default
	> 400	48	“largeinit” alternative

Note: Enlarging the MVFS caches reduces the amount of memory available to UNIX applications. If you use the “largeinit” MVFS configuration, you should also reconfigure each view that is used to access ClearCase data on that host, increasing its *view_server* cache size to 1Mb. See “Reconfiguring a View” on page 140. ♦

To change the MVFS cache sizes, perform one of the changes described below.

Selecting Alternative Cache Size Defaults—SunOS 4 Only

This technique is mutually exclusive with the technique for modifying the virtual file system table, which is described in the *ClearCase Notebook*. Exactly *one* of the *modload* commands in the ClearCase startup script must be enabled; all others must be commented out.

You can revise the ClearCase startup script, */etc/rc.atria*, to configure larger default sizes for the MVFS caches:

- MVFS-internal identifiers (*mnodes*) cached for up to 4096 MVFS objects
- Up to 1800 unused *mnode* numbers cached
- UNIX-internal identifiers (*vnodes*) cached for up to 200-1000 cleartext files, depending on the system-wide maximum number of users (MAXUSERS kernel configuration parameter)
- Up to 2800 names of MVFS objects cached

The larger caches add about 500Kb to the size of kernel (unpageable) memory, but provides better performance when the "working set" of objects in a build or command exceeds the default cache allocations.

Use the following procedure to configure the larger default caches:

1. **Shut down ClearCase.**

   ```
   # /etc/rc.atria stop
   ```

2. **Revise the cache configuration**—In the "customer-editable section", uncomment the *CONFIG 2* entry, and make sure that all other entries are commented out.

   ```
   # CONFIG 2: Configure larger caches for MVFS file system
   #
       ENTRY="-entry _xxxlargeinit"
   #
   # CONFIG 3 (DEFAULT): Use 'TFS' slot if no available VFS switch
   # entry
   #
   # TFS must not be used by any application on your host
   #
       ENTRY=""
   ```

3. **Restart ClearCase.**

   ```
   # /etc/rc.atria start
   ```

Compiling New Cache Sizes into the MVFS

You can customize cache sizes on SunOS 4 hosts or SunOS 5 hosts by recompiling the MVFS module that modload incorporates into the UNIX kernel.

Table 13-2 lists the cache parameters, with default values and suggested "larger-than-default" values. But before proceeding, be sure you will avoid the following pitfalls:

- An *mvfs_cvpfreemax* value that exceeds the recommended maximum may cause *inode table overflow* errors (reported on the system console) and/or system hangs.
- The *mvfs_mnmax* value *must* exceed the *mvfs_vobfreemax* value. We recommend that the value be about twice as large.
- Larger MAXUSERS values cause increased operating system memory utilization.

Table 13-2 Cache Parameters for MVFS module: 'mvfs.o'

MVFS Cache Parameter	Description	Default Value	Suggested Increased Value
mvfs_mnmax	system wide maximum number of mnodes	4096	4096
mvfs_vobfreemax	maximum number of objects to cache (400 bytes/object)	900	1800

Table 13-2 (continued) Cache Parameters for MVFS module: 'mvfs.o'

MVFS Cache Parameter	Description	Default Value		Suggested Increased Value	
mvfs_cvpfreemax	maximum number of cleartext files to cache	MAXUSERS	Max #	MAX-USERS	Max #
		16	100	16	250
		32	200	32	500
		48	300	48	750
		64	400	64	1000
		l*arger values: linear scaleup*		*larger values: linear scaleup*	
mvfs_dncdirmax	directory names to cache (100 bytes/entry)	200		400	
mvfs_dncregmax	regular file names to cache (100 bytes/entry)	800		1600	
mvfs_dncnoentmax	names that produce ENOENT returns (100 bytes/entry)	400		800	

SunOS 4 Cache Override Procedure

Use this procedure to customize cache sizes on hosts running SunOS 4.

1. **Gather your tools**—Make sure that the standard UNIX programs *make*(1), *cc*(1), and *ld*(1) are available on your host.
2. **Become the *root* user.**

   ```
   % su
   Password: <enter root password>
   ```

3. **Shut down ClearCase.**

   ```
   # /etc/rc.atria stop
   ```

4. **Edit the MVFS configuration file**—Edit file */usr/atria/sun4–4.n/kvm/mvfs_param.c*.

5. **Revise cache configuration parameters**—Change one or more of the MVFS cache parameters listed in Table 13-2. (Other parameters in *mvfs_param.c* generally have no effect on ClearCase performance.)

   For example, this code implements the "larger-than-default" values in the table:

   ```
   int mvfs_mnmax = 4096;
   int mvfs_vobfreemax = 1800;
   int mvfs_cvpfreemax = 300;
   int mvfs_dncdirmax = 400;
   int mvfs_dncregmax = 1600;
   int mvfs_dncnoentmax = 800;
   ```

6. **Save the MVFS configuration file.** Save the file and exit the text editor.

7. **Rebuild the *mvfs.o* file.**

   ```
   # cd /usr/atria/sun4-4.n/kvm
   # make -f config.mk
   # /etc/rc.atria start
   ```

8. **Restart ClearCase.**

   ```
   # /etc/rc.atria start
   ```

SunOS 5 Cache Override Procedure

Use this procedure to customize cache sizes on hosts running SunOS 5:

1. **Become the *root* user.**

   ```
   % su
   Password: <enter root password>
   ```

2. **Shut down ClearCase.**

   ```
   # /etc/init.d/atria stop
   ```

3. **Edit file */etc/system***—You can make the change in either, but not both, of the following ways:

   - Add this line:

     ```
     set mvfs:mvfs_largeinit = 1
     ```

   - For one or more of the MVFS cache parameters listed in Table 13-2 in "Cache Parameters for MVFS module: 'mvfs.o'" on page 137, create an entry of the form:

     ```
     set mvfs:parameter = value
     ```

For example, you might establish the following parameter settings to increase cache sizes:

```
set mvfs:mvfs_mnmax=4096
set mvfs:mvfs_vobfreemax=1800
set mvfs:mvfs_cvpfreemax=300
set mvfs:mvfs_dncdirmax=400
set mvfs:mvfs_dncregmax=1600
set mvfs:mvfs_dncnoentmax=800
```

4. **Save the /etc/system file.**
5. **Restart the operating system**—Use *reboot*(1M) or any other standard means to restart the operating system.

Reconfiguring a View

To speed its performance, the *view_server* process associated with a view maintains a cache. The default size is 204800 bytes (200Kb). You can configure a larger cache size, in order to boost performance. This is particularly useful for views in which very large software systems are built by *clearmake*.

Follow these steps to reconfigure a *view_server*'s cache:

1. **Add or revise a '-cache' line in the view's configuration file**—This is file *.view* in the view storage directory. For example:

   ```
   -cache 1048576
   ```

2. **Kill the view_server process**—On the host where the view storage directory resides, search the process table for a *view_server* that was invoked with the pathname of the view storage directory. For example:

   ```
   % cleartool lsview akp
   * akp             /net/neon//home/hui/views/akp.vws
   % ps -ax | grep 'view_server.*akp.vws'
   5011 ... view_server /net/neon/home/akp/views/akp.vws
   % kill 5011
   ```

3. **Restart the view_server process**—Use a *startview* or *setview* command:

   ```
   % cleartool startview akp
   ```

Chapter 14

Making a VOB or View Inaccessible

This chapter presents procedures for rendering a VOB or view temporarily inaccessible to all client processes. ClearCase's central *storage registry* makes this easy.

Alternative to VOB Deactivation

In some situations, you may not need to make the data totally inaccessible. For example, if you merely wish to prevent a VOB from being modified, you can *lock* it with having to take it "off-line":

```
% cleartool lock -vob pathname
```

To lock an entire VOB, you must be either *root* or the VOB owner. The *pathname* you specify can be either the VOB storage directory or the VOB-tag (mount point), or any pathname under the VOB-tag. In the latter case, you must enter the command in a view context.

Taking a VOB Out of Service

Suppose that the VOB to be taken out of service has storage directory */net/sol/vobstore/libpub.vbs*, and has VOB-tag */proj/libpub*. Follow these steps to make the VOB inaccessible:

1. **Have all clients deactivate the VOB**—On each client, this command unmounts the VOB:

   ```
   % cleartool umount /proj/libpub
   ```

   (ClearCase Release 1 client hosts must use the system-level *umount*(1M) command.)

2. **Remove the VOB from the object registry**—This will prevent anyone from reactivating the VOB:

   ```
   % cleartool unregister -vob /net/sol/vobstore/libpub.vbs
   ```

3. **(optional) Remove the VOB from the tag registry**—If you leave the VOB-tag for this VOB in the tags registry, attempts to mount the VOB will produce an unattractive message:

   ```
   % cleartool mount /proj/libpub
   cleartool: Error: An error occurred mounting.
   Refer to the log file
   "/usr/adm/atria/log/mntrpc_server_log"
   for more information on the warning or failure.
   ```

   This message may be tolerable—even desirable. But if you would prefer a less verbose message, remove the VOB-tag:

   ```
   % cleartool rmtag -all -vob /proj/libpub
   % cleartool mount /proj/libpub
   cleartool: Error: /proj/libpub is not a registered vob tag.
   ```

   (The *-all* option ensures correctness in a network with multiple regions—the VOB-tag is removed from all the logically-distinct tags registries.)

4. **If the VOB has ClearCase Release 1 clients, rename the VOB storage directory**—Release 1 clients do not use the object registry to determine the location of the VOB storage directory. Step #2 does not "hide" the VOB from such clients, so you must move its storage directory aside. For example:

   ```
   % rlogin sol -l root
   Password: <enter password>
   % mv /vobstore/libpub.vbs /vobstore/libpub.vbs.NOACCESS
   % exit
   ```

5. **Terminate the VOB's server processes**—Search the process table of the VOB host for the ClearCase *vob_server* and *vobrpc_server* processes that manage that VOB. Use `ps -ax` or `ps -ef`, and search for "/vobstore/libpub.vbs"; use *kill*(1) to terminate any such processes. (Only the *root* user can kill a *vobrpc_server* process.)

Restoring the VOB to Service

When the VOB is to be restored to service, follow these steps:

1. **(if necessary) Rename the VOB storage directory**—If you moved the VOB storage directory aside in Step #4 above, move it back:

   ```
   % rlogin sol -l root
   Password: <enter password>
   % mv /vobstore/libpub.vbs.NOACCESS /vobstore/libpub.vbs
   % exit
   ```

2. **Restore the VOB to the object registry.**

   ```
   % cleartool register -vob /net/sol/vobstore/libpub.vbs
   ```

3. **(if necessary) Restore the VOB to the tag registry**—This is necessary only if you removed the VOB-tag in Step #3 above:

   ```
   % cleartool mktag -vob -tag /proj/libpub \
       /net/sol/vobstore/libpub.vbs
   ```

   (Repeat, as necessary, for other network regions.)

4. **Have clients reactivate the VOB**—On each client host, this command mounts the VOB:

   ```
   % cleartool mount /proj/libpub
   ```

   (ClearCase Release 1 client hosts must use the system-level *mount*(1M) command.)

Taking a View Out of Service

The procedure for taking a view out of service temporarily is essentially equivalent to the VOB procedure described in "Taking a VOB Out of Service" on page 141. The salient differences are:

- There is no specific "unmount" command for views. In particular, there is no command that removes a view-tag from a client hosts' viewroot directory.
- The only view-related server process to be terminated is the *view_server*, which runs on the host where the view storage directory resides.

Restoring the View to Service

The procedure for restoring a view to service is essentially similar to that described in "Restoring the VOB to Service" on page 143.

Permanent Removal of a VOB or View

Use the *rmvob* command to remove a VOB permanently; use *rmview* to remove a view permanently. These command automatically remove all relevant entries from the ClearCase storage registry.

Chapter 15

Determining a Data Container's Location

This chapter demonstrates how to determine the actual storage locations of *MVFS files*—those accessed through VOB directories. Both standard UNIX tools and the ClearCase *mvfsstorage* utility are used.

Scenario

Let's focus on three files within VOB directory */proj/monet/src*, as seen through view *allison_vu*:

- Element *cmd.c* has element type *text_file*, and is currently checked-out.
- Element *monet.icon* has element type *file*, and is not currently checked-out.
- File *ralph_msg* is a view-private file, created by saving an electronic mail message to disk.

Determining the ClearCase Status of Files

The *describe* command verifies that the three files are as described above:

```
% cleartool describe cmd.c monet.icon ralph_msg
version "cmd.c@@/main/CHECKEDOUT" from /main/6 (reserved)
  checked out 03-Feb-93.20:40:30 by (allison.mon@phobos)
  by view: "phobos:/usr/people/arb/view_store/arb.vws"
  element type: text_file

version "monet.icon@@/main/1"
  created 03-Feb-93.20:17:04 by (allison.mon@phobos)
  element type: file

View private file "ralph_msg"
  Modified: Wednesday 02/03/93 21:39:49
```

Determining the Full UNIX Pathnames of Files

The standard *ls*(1) and *pwd*(1) commands show the full pathnames of the files, from UNIX's point of view:

```
% ls -l 'pwd'/cmd.cex
-rw-rw-r--   1 allison  mon     211 Feb  2 12:03
/proj/monet/src/cmd.c

% ls -l 'pwd'/monet.icon
-r--r--r--   1 allison  mon     266 Feb  3 20:17
/proj/monet/src/monet.icon

% ls -l 'pwd'/ralph_msg
-rw-rw-r--   1 allison  mon     852 Feb  3 20:40
/proj/monet/src/ralph_msg
```

Where is the VOB?

The UNIX *ls* command provides no clue as to where the VOB is mounted. It might be mounted at */proj*, or at */proj/monet*, or at */proj/monet/src*. The *describe* command provides the answer:

```
% cleartool describe -vob /proj/monet/src
versioned object base "/proj/monet"
  created 01-Feb-93.17:35:03 by (vobadm.vobadm@sol)
  VOB storage remote host:path
    "sol:/usr/vobstore/monet.vbs"
  VOB storage local pathname "/net/sol/vobstore/monet.vbs"
  VOB ownership:
    owner vobadm
    group vobadm
```

This command shows that the VOB is mounted at */proj/monet*. In addition, it shows the local pathname to the VOB storage area. This looks like an *automount*(1M) pathname:

`/net/sol/`	`vobstore/monet.vbs`
(storage area is located on host sol)	*(pathname of VOB storage area on host sol)*

To make sure, another *mount* command verifies the location of the VOB storage area:

```
% /etc/mount | grep vobstore
  .
  .
sol:/usr/vobstore on /tmp_mnt/net/sol/vobstore type nfs ...
```

This command shows explicitly that the VOB storage area is located with directory */usr/vobstore* on host *sol*.

Where is the View?

The *pwv* ("print working view") and *lstag* ("list view-tag") commands show the location of the view storage area:

```
% cleartool pwv
Working directory view: allison_vu
Set view: allison_vu

% cleartool lstag allison_vu
* allison_vu /net/phobos/usr/people/arb/view_store/arb.vws
```

As in the preceding section, you might need to use another *mount* command to identify the host on which the view storage area resides ("which host contains directory */net/phobos*"?).

Where are the Individual Files?

The *data containers* for all MVFS files are *logically* stored within a VOB or view storage area, as shown in Table 15-1.

Table 15-1 Storage Locations of MVFS Files

Kind of File	Storage Location
version (checked-in)	VOB *source* storage pool (and perhaps VOB *cleartext* storage pool, also)
checked-out version	view-private data storage

Table 15-1 (continued) Storage Locations of MVFS Files

Kind of File	Storage Location
unshared derived object	view-private data storage
shared derived object	VOB derived object storage pool
view-private file	view-private data storage

The following sections show how the *mvfsstorage* utility indicates the exact storage location of an MVFS file. This utility is located in directory */usr/atria/etc*, a directory that is not typically on your search path. Thus, you may need to invoke *mvfsstorage* using a full pathname, as in the examples below.

Locating a Checked-Out Version

mvfsstorage shows the location in view-private data storage of the checked-out version of *text_file* element *cmd.c*:

```
% /usr/atria/etc/mvfsstorage cmd.c
/net/phobos/usr/people/arb/view_store/arb.vws/.s/
80000002.VOB/8000000B.00B0.cmd.c
```

Locating a Checked-In Version's Cleartext Container

For a checked-in version of a *text_file* element, *mvfsstorage* shows the location of the cleartext data container into which the type manager extracts the version:

```
% /usr/atria/etc/mvfsstorage cmd.c@@/main/1
/net/sol/vobstore/monet.vbs/c/cdft/28/32/
8a1a9a50010e11cca2ca080069021822

% /usr/atria/etc/mvfsstorage cmd.c@@/main/2
/net/sol/vobstore/monet.vbs/c/cdft/3a/33/
8e4a9a54010e11cca2ca080069021822
```

Locating a Checked-In Version's Source Container

For a checked-in version of a *file* element, *mvfsstorage* shows the location of the data container in the source pool:

```
% /usr/atria/etc/mvfsstorage monet.icon
/net/sol/vobstore/monet.vbs/s/sdft/26/4/
474fa2f4021e11cca42f0800690605d8
```

ClearCase does not maintain cleartext versions of *file* elements, or any other element type for which the cleartext of a version is stored in its source pool data container. Instead, programs access the data container in the source pool directly.

Locating a View-Private File

Like a checked-out version, a view-private file is located in a view's private data storage:

```
% /usr/atria/etc/mvfsstorage ralph_msg
/usr/people/arb/view_store/arb.vws/.s/
80000002.VOB/8000000C.00BD.ralph_msg
```

Non-Local Storage

The *mvfsstorage* utility shows the logical location of data containers within VOB and view storage directories. But VOB storage pools can be located outside the VOB storage directory itself; likewise, a view's private storage area can be located outside the view storage directory.

If *mvfsstorage* indicates that a data container is in a non-default VOB storage pool, use the *lspool* command to determine the location of the pool. The default pools are *sdft* (default source pool), *cdft* (default cleartext pool), and *ddft* (default derived object pool). For example:

```
% /usr/atria/etc/mvfsstorage hello.h
/vobstore/monet.vbs/c/clrtxt.1/36/f/6b6ed22b08da11cca0ef0800690605d8
     (clrtxt.1 is a non-default cleartext pool)
% cleartool lspool -l clrtxt.1
pool "clrtxt.1"
 08-Feb-93.10:25:46 by (vobadm.vobadm@starfield)
  "nonlocal cleartext storage for monet VOB"
  kind: cleartext pool
  pool storage remote host:path "sol:/netwide/public/ccase_pools/clrtxt.1"
  pool storage local pathname "/vobstore/monet.vbs/c/clrtxt.1"
  maximum size: 0  reclaim size: 0  age: 96
```

Use UNIX *ls* to determine whether a view's private storage area (subdirectory *.s*) is nonlocal:

```
% ls -ld ~jones/view_store/temp.vws/.s
lrwxrwxr-x   1 jones    dvt       34 Feb 17 17:06
/usr/people/jones/view_store/temp.vws/.s -> /public/jones_temp
```

Links and Directories

The discussion above omitted links and directories. Briefly:

- For a link, *mvfsstorage* indicates the storage location of the object to which the link points. This applies to all links: view-private hard links and symbolic links, VOB hard links, and VOB symbolic links.
- A view-private directory does not have a data container; nor does a directory element. In both cases, *mvfsstorage* simply echoes the directory name.

Chapter 16

Adjusting the ClearCase Startup/Shutdown Script

This chapter describes changes you can make to the ClearCase startup/shutdown script on any host where ClearCase is installed.

Name of Startup/Shutdown Script

The name of the ClearCase startup/shutdown script is architecture-dependent:

/etc/rc.atria	SunOS-4, HPUX-9
/etc/init.d/atria	SunOS-5, IRIX-5
/sbin/init.d/atria	OSF/1-V2

Changing VOB Mounts to "Release 1 Style"

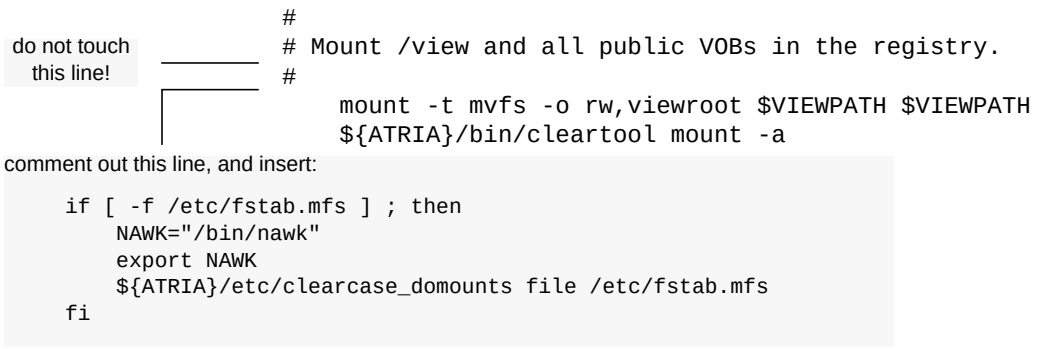
One of the principal jobs of the ClearCase startup script is to mount VOBs as *multiversion file systems* (file system type MVFS). The way in which this is accomplished changed between ClearCase Release 1 and Release 2:

- In Release 1, VOBs to be mounted by the startup script were listed in a ClearCase-specific file system table, implemented as file */etc/fstab.mfs*. (This file could, in turn, incorporate the contents of an NIS map.)
- In Release 2, all information necessary for mounting a VOB is stored in the network-wide *storage registry*. The *fstab.mfs* file is no longer required.

For compatibility, the Release 1 mechanism for mounting VOBs is still supported in Release 2 (though the file system type has been changed from "mfs" to "mvfs"). If a host was upgraded from Release 1 to Release 2, and

you wish to have it continue to mount VOBs using the old mechanism, follow these steps:

1. **Edit the ClearCase startup script**—This script's location and contents varies from architecture to architecture .



   The first command—the one you should not touch—mounts the *viewroot* directory, */view,* directly. In ClearCase Release 1, a file system table entry specified the mounting of this directory.

2. **Verify that there is no** *viewroot* **mount entry**—ClearCase installation should have commented out any */view* entry in your host's standard file system table (for example, */etc/fstab* or */etc/vfstab*). Such entries reflect ClearCase Release 1.1.x functionality that is *not* supported by Release 2. If necessary, remove such an entry manually.

3. **Revise the ClearCase file system table**—Edit file */etc/fstab.mfs* as follows:

   - Make sure it does not include any Release-1-style entries for mounting the viewroot directory.
   - Change all occurrences of the file system type *mfs* to *mvfs*.

Changing Dynamic Loading of the MVFS

Note: *This section applies to SunOS 4 hosts only.* ♦

The ClearCase startup script has a "customer-editable section", which is largely self-documenting. This section allows you to select among several invocations of the *modload*(8) utility, which dynamically loads the ClearCase MVFS code into the UNIX kernel.

The default *modload* invocation causes the MVFS to take the "TFS" slot in the virtual file system table (if no empty slot is available), and to use standard cache sizes. You can select either of these alternative invocations:

- **Use the "RFS" slot instead of the "TFS" slot**—For an additional discussion, see the *CASEVision/ClearCase Release Notes*.
- **Configure larger MVFS caches**—This is a performance-tuning measure. For an additional discussion, see "Improving Client Host Performance" on page 133.

The sections cited above also discuss other techniques for manipulating the virtual file system table, and for configuring MVFS caches.

Chapter 17

Adjusting ClearCase License Information

This manual contains a place-holder chapter on adjusting license information to maintain consistency with ClearCase documentation for other platforms. This subject is not relevant for the Silicon Graphics implementation of ClearCase.

On Silicon Graphics platforms, ClearCase licensing is handled through NetLS. Refer to Chapter 7 of the *CASEVision/ClearCase Release Notes* to learn more about ClearCase licensing. Read the *NetWork License System Administration Guide* to learn more about NetLS.

Chapter 18

Adjusting ClearCase Registry Information

After a brief review of ClearCase storage registry concepts and terminology, this chapter describes the procedure for adding a new network region. It concludes with a set of general guidelines for registry-related administration.

Registry Review

At this point, you should be familiar with the ClearCase *storage registry* and, possibly, with *network regions* as well. Chapter 3, "Network-Wide Access to ClearCase Data", introduces the ClearCase storage registry, including the VOB and view *tag* and *object* registry files. Chapter 3 also describes network regions, including a sample network with two regions, *uno* and *dos*.

Chapter 6, "Setting Up ClearCase VOBs", describes the process of creating and registering a VOB. It introduces the subject, which this chapter expands on, of using multiple network regions to provide global network access to VOB storage directories.

Later, Chapter 19, "Changing the Location of Network-Wide Resources", describes the procedures for moving the ClearCase storage registry and for renaming the registry server host.

ClearCase Storage Registry

All VOBs and views are registered in the *ClearCase storage registry*. The storage registry resides on the *registry server host*, in */usr/adm/atria/rgy*, and it includes two distinct pieces, a *VOB registry* and a *view registry*. The VOB registry and view registry each include two files: one registers tags, the other registers storage directories. The files are *vob_tag*, *vob_object*, *view_tag*, and *view_object*.

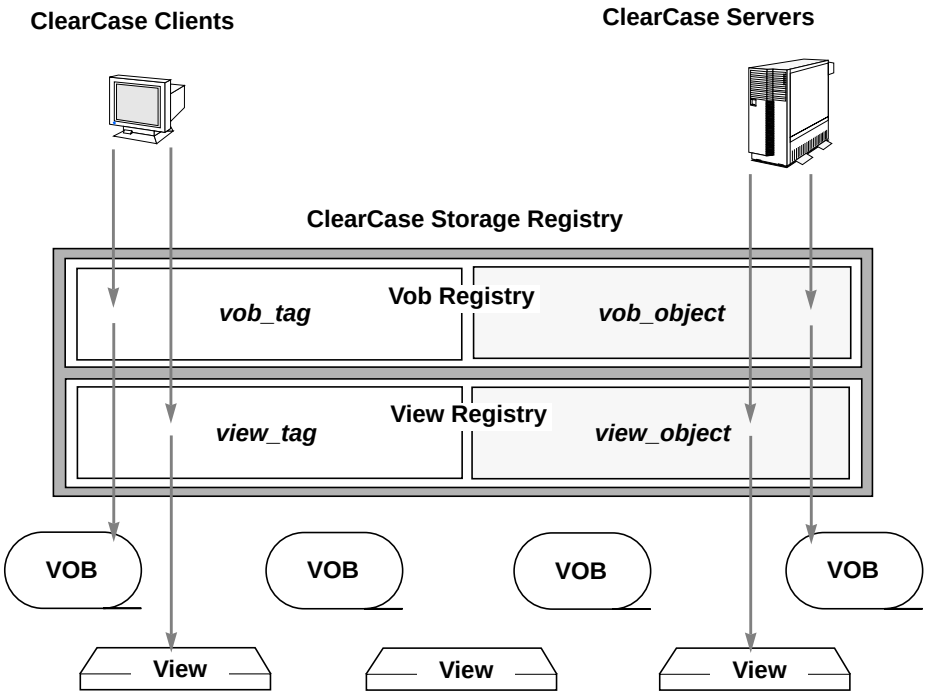


Figure 18-1 ClearCase Storage Registry

Object Registry Files

The *vob_object* and *view_object* registry files record the location of each VOB and view using a *host-local* pathname. That is, the pathname to the data structure is one that is valid on the host where the storage directory resides. These pathnames are used by the ClearCase server processes (*view_server*, *vob_server*, and so on), which always run on the applicable VOB or view storage host.

Tag Registry Files

For most purposes, client processes access VOBs and views via *VOB-tags* and *view-tags*, not physical storage locations. These logical locations are resolved to physical storage locations through lookups in the *vob_tag* and *view_tag* registry files. Each tag registry entry includes a *global pathname* to

the corresponding VOB or view storage directory—a pathname that is valid on all ClearCase client hosts in the network region to which the tag applies.

Three points deserve special emphasis:

- A tag entry's global access pathname must be valid on all hosts in the tag's network region, including the corresponding storage directory's host.
- For all practical purposes, developers cannot access a VOB or view unless it has both an object registry entry and a tag registry entry. (An administrator can perform some specialized operations on a VOB or view storage directory that has no corresponding tag.)
- If your network has more than one network region, a VOB or view needs a separate tag entry (but not a separate tag *name*) for each region with hosts that will access the VOB or view.

cleartool Commands and Registry Files

The bullet items from the previous subsection have several practical implications for the system administrator. In particular:

- Be "on the lookout" for faulty global access pathnames stored in the two tag registry files.
- If a VOB or view access operation fails, first make sure a VOB-tag or view-tag exists for the network region from which the access is attempted. (A common mistake is attempting to access a VOB or view that has no tag.)

lsvob and *lsview (-long)*

Use *lsvob* or *lsview* with the `-long` option to report the key registry information. If you cannot "see" a VOB with *lsvob*, or a view with *lsview*, then it does not have a tag registered for the current network region (or for the region specified in the command). For a VOB or view with a registered tag, the output from *lsvob* `-long` and `lsview -long` includes:

- the VOB-tag or view-tag
- the host name for the host on which the storage directory resides

- the "host-local" access pathname (from the *vob_object* or *view_object* file), which the relevant VOB or view server uses to access the storage directory
- the global pathname (from the *vob_tag* or *view_tag* file), which is used to resolve references to the newly created tag

For example:

```
% cleartool lsvob -long
  .
  .
Tag: /vobs/src
  Global path: /net/venus/vobstore/src.vbs
  Server host: venus
  .
  .
  Vob server access path: /vobstore/src.vbs
  .
  .
% cleartool lsview -long
  .
  .
Tag: main
  Global path: /net/venus/viewstore/main.vws
  Server host: venus
  .
  .
  View server access path: /viewstore/main.vws
  .
  .
```

If the global and server access paths are identical, or if they don't look right to you, then you might anticipate problem reports regarding access to the VOB or view.

mkvob and *mkview*

The *mkvob* and *mkview* commands create both the object and tag registry entries necessary for ClearCase client access. (There is one exception: if you create a VOB with a public tag, but the tag password fails, the VOB is created without the tag. You must create the tag in a separate operation with *mktag*.) Like *lsvob* and *lsview*, *mkvob* and *mkview* output includes:

- the “host-local” access pathname (from the *vob_object* or *view_object* file)
- the global pathname (from the *vob_tag* or *view_tag* file)

Sample *mkvob* output:

```
  .
  .
Host-local path: venus:/vobstore/src.vbs
Global path:     /net/venus/vobstore/src.vbs
  .
  .
```

Sample *mkview* output:

```
  .
  .
Host-local path: venus:/viewstore/main.vws
Global path:     /net/venus/viewstore/main.vws
  .
  .
```

mktag and *rmtag*

Use *mktag* to add or replace tag entries for existing VOBs and views. Two common uses are:

- creating additional VOB-tags and view-tags to support multiple network regions
- converting a private VOB to a public VOB

To change a tag’s name, or to change its assigned network region, use *rmtag* and *mktag* in succession, not `mktag -replace`. See the *mktag* and *rmtag* manual pages for more detail on tag creation and removal.

Figure 18-2 shows all of the *cleartool* commands that affect registry files. See also the *registry_ccase* manual page.

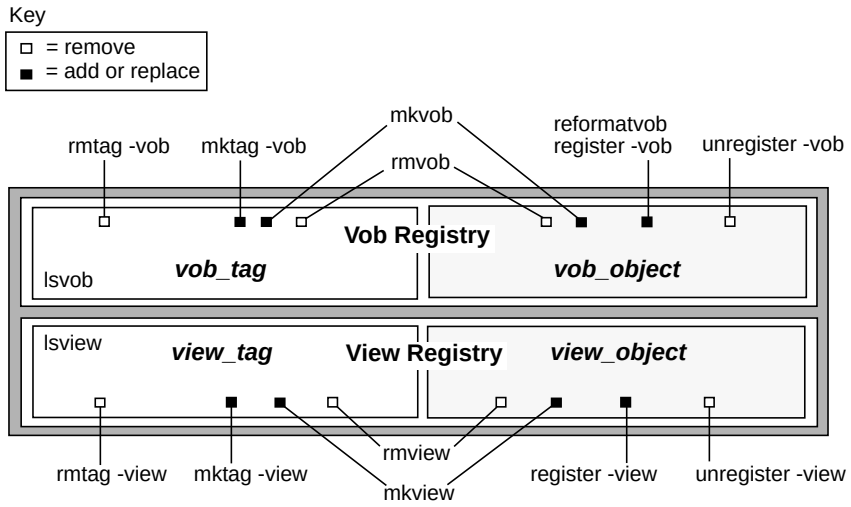


Figure 18-2 *cleartool* Commands and the ClearCase Storage Registry

Adding a Network Region

The section "Ensuring the VOB's Global Accessibility" on page 65 outlines the circumstances which may require you to adjust the automatically generated registry entry for a VOB or view. That section goes on to present some sample *mktag* commands that "fix" the storage registry by explicitly specifying, for a VOB or view, the host-local and global access paths. This section applies in the case where a single, global pathname to a VOB or view storage directory does not exist for all network hosts that must access it—so you must partition your network into at least two *network regions*. If you have not read the section "Network Regions" on page 28, please do so before proceeding.

When to Partition the Network

Of the network configurations that demand multiple network regions, two are most common:

- **Multiple network interfaces to a VOB or view host**—For example, a host might be known to some hosts (and their automounter programs) as *neptune*, and to other hosts as *neptune-gw*. In this case, the same VOB might have two "global" storage pathnames:

  ```
  /net/neptune/vobstore/src.vbs
  /net/neptune-gw/vobstore/src.vbs
  ```

- **Multiple host architectures in the same network**—For example, a view that is accessed as */net/neptune/viewstore/main.vws* on a UNIX host may be accessed as *V:\viewstore\main.vws* on a Windows/NT host. (Note that when pathnames differ *syntactically* between hosts, it is not just storage access paths that are guaranteed to differ, but VOB-tags as well.)

A Sample Network Partition

Figure 18-3 illustrates a simple two-region network. Two network regions are required because a (shared) VOB host has two network interfaces and is known to some hosts as *neptune*, and as *neptune-gw* to others. Although hosts in both regions use the same VOB-tags (and view-tags), they access VOB/view storage—at least on *neptune*/*neptune-gw*—using different pathnames. Therefore, separate tag registry entries are required.

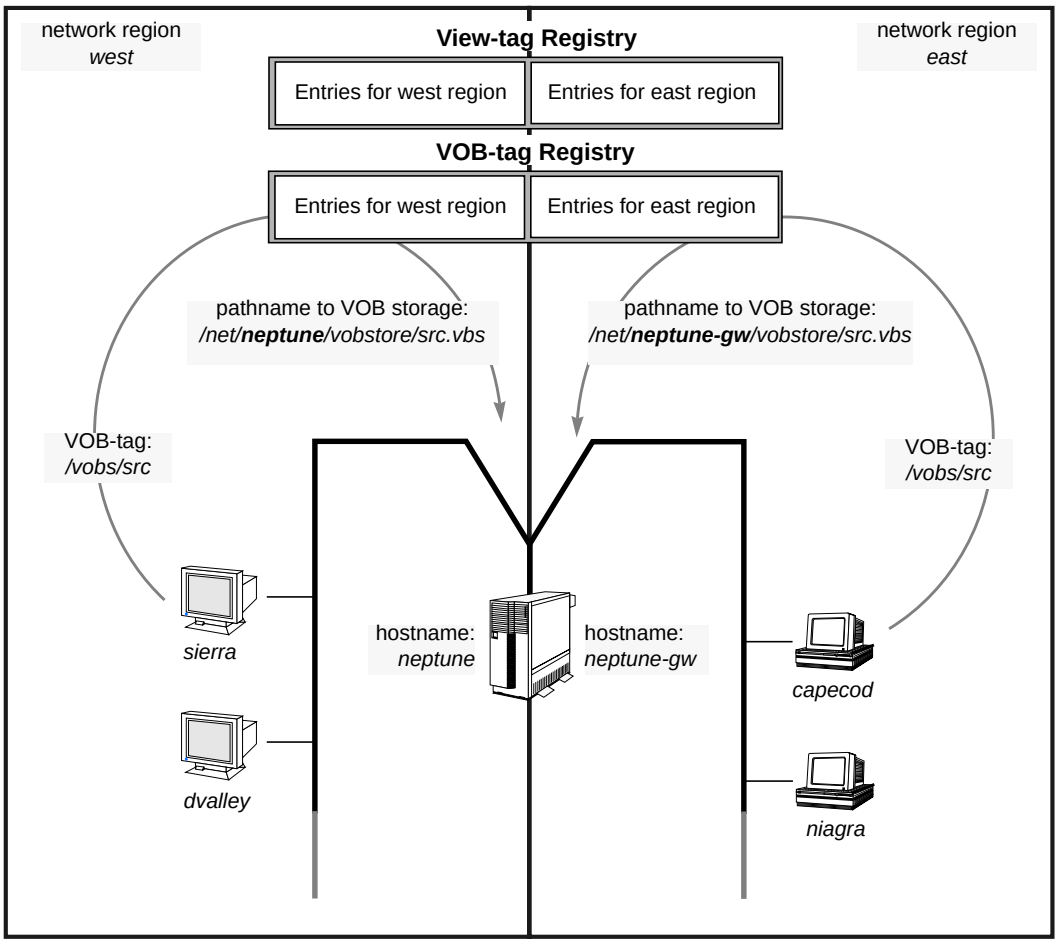


Figure 18-3 Sample Network with Two Regions

Procedure for Adding a Network Region

Sample commands in this procedure use host and pathnames from the network in Figure 18-3.

Note: If you are a new ClearCase installation, and you are partitioning the network into regions from the start, then use this procedure to add regions 2 through *n* after (1) ClearCase has been installed on each host, and (2) shared VOBs and views have been created on the network's VOB and view storage hosts. ♦

1. **Pick a region name**—In this example, we use *west* as the region name.
2. **Determine the "global" access paths for the new region**—For each shared VOB and view, determine a network address to the storage directory that is valid on *every* host that will be in the new region. (You may need to take into account a nonstandard automount program or other local mounting policies.) For example, on *sierra*:

   ```
   % ls -d /net/neptune-gw/vobstore/src.vbs
   /net/neptune-gw/vobstore/src.vbs
   % ls -d /net/saturn/viewstore/view1.vws
   /net/saturn/viewstore/view1.vws
   % ls -d /net/neptune/vobstore/incl.vbs
   ... No such file or directory
       (no good; can't see neptune...)
   % ls -d /net/neptune-gw/vobstore/incl.vbs
       ( ...try again with neptune-gw)
   /net/neptune-gw/vobstore/incl.vbs
   % ls -d /net/dvalley/viewstore/view3.vws
       (storage on host in west region)
   /net/dvalley/viewstore/view3.vws
     .
     .
   ```

   For each VOB or view storage directory, make sure the *same* pathname "works" from *all* hosts in the new region.
3. **Move some hosts to the new region**—Use the procedure in "Moving a Host to a New Network Region" on page 168.
4. **Log in to any host in the new network region**—You need not be *root*, but you should know the network's *VOB-tag password*, in order to create public VOB-tags in the new region.

   ```
   % rlogin sierra
   ```

5. **`Create` VOB-tags for the new region**—Each VOB can have at most one tag in a region. In a typical network with ***n*** regions, each VOB or view storage directory has ***n*** tag entries.

   Note: A VOB or view need not have a tag in every region. However, a VOB or view is inaccessible for development work on hosts in any region for which it is "tagless". This suggests that you might use network regions as "access domains" instead of "naming domains". ♦

   For a VOB or view to be visible in multiple network regions, it must have a separate tag *entry* for each region. However, the actual tag *name* (*/vobs/src* or *proj2_main_view*, for example) should be the same for all regions. The ideal is consistent, transparent access from any network region; this is possible only if the tag itself is constant across regions.

   For example:

   ```
   % cleartool mktag -vob -tag /vobs/src \
       /net/neptune-gw/vobstore/src.vbs
   % cleartool mktag -vob -tag /vobs/incl \
       /net/earth/vobstore/incl.vbs
     .
     .
   ```

   This command creates an entry for VOB */net/neptune-gw/vobstore/src.vbs*, for region *west*, in the *vob_tag* registry file. (Because we are *in* the target region, the `-region` argument to *mktag* is not required.)

   Note: The *mktag* command may fail if the VOB does not currently have a tag entry for its storage host's network region. ♦

6. **Verify the new VOB-tags**—On a host in the *west* region:

   ```
   % cleartool lsvob -long
   ```

   On a host in a different region:

   ```
   % cleartool lsvob -long -region west
   ```

   To verify the correctness of your work in Step #5, mount each shared VOB from a host in the *west* region. For example:

   ```
   % cleartool mount /vobs/src
   ```

If access fails, and your network connection to the VOB storage host is good: use the guidelines in this chapter and in Chapter 6, "Ensuring the VOB's Global Accessibility" on page 65, to determine whether to supply new access path information for the VOB, using the `-host`, `-hpath`, and `-gpath` arguments to *mktag*.

7. **Create view-tags for the new region**—On a host in the *west* region, for each shared view:

   ```
   % cleartool mktag -view -tag view1 /net/saturn/viewstore/view1.vws
   % cleartool mktag -view -tag view3 /net/dvalley/viewstore/view3.vws
     .
     .
   ```

8. **Verify the new view-tags**—On a host in the *west* region:

   ```
   % cleartool lsview -long
   ```

   On a host in a different region:

   ```
   % cleartool lsview -long -region west
   ```

   To verify the correctness of your work in Step #7, activate each view from a host in the new region.

   ```
   % cleartool startview view1
   ```

   If access fails, and your network connection to the view storage host is good: use the guidelines in this chapter (and in chapters this one references) to determine whether to supply new access path information for the view, using the `-host`, `-hpath`, and `-gpath` arguments to *mktag*.

This completes the procedure for adding a new network region. From this point on, each new shared VOB or view created on the network requires a tag entry for each region. That is, if a new VOB or view is created anywhere on the network, and it must be visible to hosts in both the *east* and *west* regions, you must execute an additional *mktag* command. For example:

- After creating VOB */vobs/lib* on *neptune* in the *east* region:

  ```
  % cleartool mktag -vob -tag /vobs/lib -region west \
      /net/neptune-gw/vobstore/src2.vbs
  ```

- After creating view *view4* on host *dvalley* in the *west* region:

```
% cleartool mktag -view -tag view4 -region east \
    /net/dvalley/viewstore/view4.vws
```

Note: Following the guidelines set out in this chapter, you would log in to a host in the target region before executing the *mktag* command. This makes the *-region* option unnecessary. ♦

Moving a Host to a New Network Region

Once network regions are established, moving a host from one region to another (to balance the network load between two interface cards, for example) is straightforward. The following steps move host *niagra* to the *west* region:

1. **Verify VOB and view access paths**—Make sure that the tag registry entries for the *west* region are valid on *niagra* as well. List all VOBs and views registered in the *west* region.

```
% cleartool lsvob -long -region west
  .
  .
Tag: /vobs/src
  Global path: /net/neptune-gw/vobstore/src.vbs
  .
  .
% cleartool lsview -long -region west
  .
  .
Tag: main
  Global path: /net/saturn/viewstore/view1.vws
  .
  .
```

On *niagra*, verify the global access paths.

```
% ls -d /net/neptune-gw/vobstore/src.vbs
/net/neptune-gw/vobstore/src.vbs
% ls -d /net/saturn/viewstore/view1.vws
/net/saturn/viewstore/view1.vws
  .
  .
```

2. **Log in as *root* on the host to be moved.**
3. **Stop ClearCase processing.**

   % **`/etc/rc.atria stop`**
   (startup/shutdown command is architecture-dependent)
4. **Change the region assignment**—Put the new region name into file */usr/adm/atria/rgy/rgy_region.conf*.

   % **`echo west > /usr/adm/atria/rgy/rgy_region.conf`**
5. **Restart ClearCase processing.**

   % **`/etc/rc.atria start`**

Removing a Network Region

At some point, you may want to dissolve a network region and re-incorporate its hosts back into one or more existing regions. Move each host to its "new" region as described in the preceding paragraph. Then, delete the region's VOB-tags and view-tags with *rmtag*.

Registry-Related Guidelines

Following are some general tips, warnings, and guidelines regarding ClearCase storage registry administration:

- You cannot access a VOB or view (even to remove it) unless it has both a tag registry entry and an object registry entry. Use *lsvob* or *lsview* to see if either the tag is missing (no listing at all) or the object entry is missing (no storage paths appear in the **`-long`** output).

  If you cannot access a VOB or view, first make sure it has a tag. If it does not appear in the output of an *lsvob*/*lsview* command, it has no tag. Create one with *mktag*.

  If it has a tag, list the tag with **`lsvob -long`** or **`lsview -long`**. If the output includes incorrect pathnames (identical local and global pathnames usually mean trouble), you can try to re-register the object (with *mktag* and/or *register*) to supply correct pathnames. If everything looks OK, contact Atria Customer Support.

- Don't use full pathnames in links on VOB or view hosts. The global pathname may be unresolvable for non-local hosts. For example:

  `/usr/vobs -> /usr/vobstore`
  Wrong. When this link is resolved on a client host, the full pathname "/usr/vobstore" references the */usr* directory on the client host.

  `/usr/vobs -> ../vobstore`
  Right. When this link is resolved on a client host, the relative pathname "../vobstore" references a location on the VOB host.

  Similarly, beware of full pathnames in VOB symbolic links that point to locations in other VOBs.

- On VOB and view hosts, watch out for links to unexported disk partitions. If you cannot access a VOB and suspect a faulty registry pathname, follow the registered global pathname to the storage host and *pwd* to see where you are. Make sure a link does not point to a location in an unexported partition. For example:

  `/usr/vobs -> ../vobstore`
  This link causes failures if the location "/usr/vobstore" is in another partition which is not exported.

- Don't rename a tag; remove it and create a new one.
- Don't *rm* a storage directory; this doesn't clean up the registry entries.
- Don't delete tags as a precursor to deleting the storage directory; let *rmvob* or *rmview* delete tags for you.
- When creating a VOB or view, creating a VOB-tag or view-tag, registering or unregistering a storage directory, or reformatting a VOB—especially when using the `-host/-hpath/-gpath` triplet—execute the command on the host where the VOB or view storage directory resides. This enables ClearCase to perform some pathname validations, which it cannot perform when the command is executed from a remote host.
- Don't administer VOBs and views from a host running ClearCase Release 1.1.*x*. This will leave the storage registry in an inconsistent state. For example, deleting a VOB using ClearCase Release 1.1.*x* fails to remove *vob_tag* and *vob_object* registry entries. The deleted VOB appears in *lsvob* output, even though its storage directory no longer

exists. In addition, the VOB scrubbing utilities *scrubber* and *vob_scrubber* use the *vob_object* entry to locate the VOB. They will report errors when they fail to find the deleted storage directory.

Multiple Network Regions

- Use the same VOB-tags for all network regions. Each VOB or view that must be visible in multiple network regions must have a separate tag *entry* for each region. However, the tag *name* (*/vobs/src* or *proj2_main_view*, for example) should be the same for all regions. The goal is consistent, transparent access from any network region; this is possible only if the tag is constant across regions.
- Use consistent naming conventions for directories that hold VOB mount points, and for directories that store VOB/view storage directories.
- Create VOBs and views, VOB-tags and view-tags, from a host in the network region to which the tag applies.
- Don't remote VOB storage pools for a VOB that must be accessed by hosts in more than one network region. The remote storage pool pathname must apply to all hosts that will access the VOB, but the existence of multiple network regions implies that different sets of pathnames are required to access VOBs from different hosts in the network. To repeat, *if you use a remote storage pool for a shared VOB, the pool must be accessible from all hosts, in all regions, using the same global pathname.*
- You can isolate an arbitrary group of hosts with a separate network region, or by setting up a separate registry host with its own registry files and assigning the desired hosts to the new registry. It is more common to use a network region for this purpose, as reassigning hosts to a new registry renders *all* VOBs and views in the pre-existing registry invisible to the reassigned hosts.
- A VOB's first tag (the one created by *mkvob*) must be in the "home region"—the region in which the VOB host resides. The same restriction applies to a view's first tag. You can then create additional tags for other regions, but make sure that a "home region" tag always exists.

Changing the Location of Network-Wide Resources

This chapter describes procedures for changing which hosts provide ClearCase network-wide resources and services. This includes the ClearCase release area, the license server, and the registry server. It also includes the renaming of server hosts.

The special cases of moving VOBs and views are not discussed in this chapter; these cases are covered in Chapter 11 and Chapter 12.

Changing the Location of the Release Area

To change the location of the ClearCase network-wide release area, follow these steps:

1. **Reload the distribution medium**—Create a new release area, using the procedure in the *ClearCase Notebook*.
2. **Reinstall hosts, as appropriate**—Reinstall any host whose previous installation involved one or more symbolic links to the network-wide release area. (The default installation mode copies some files and links others.) Use the procedure in the *ClearCase Notebook*.
3. **Remove the old release area**—When all hosts have been reinstalled, you can remove the old release area.

Renaming the Release Host

If you rename the network-wide release host, reinstall any host whose previous installation involved one or more symbolic links to the network-wide release area.

Moving Licenses to Another Host and Renaming a License Server Host

On Silicon Graphics platforms, ClearCase licensing is handled through NetLS. Refer to Chapter 7 of the *CASEVision/ClearCase Release Notes* to learn more about ClearCase licensing. Read the *NetWork License System Administration Guide* to learn more about NetLS.

Moving the ClearCase Registry

Use this procedure to move the ClearCase storage registry directory tree to another host.

1. **Copy the entire 'rgy' directory tree to the destination host**—Perform the copy as *root*, and make sure that *root* has remote-access rights on the destination host. For example:
   ```
   # cd /usr/adm/atria
   # tar -cf - rgy | rsh ccsvr04 'cd /usr/adm/atria ; tar -xBpf -'
   ```
   (-B option is not necessar,and not supported if remote host is an HP-UX system)

```
# exit
%
```

2. **Reconfigure the "old" host not to be a registry server host**—Remove the file that identifies the host as a registry server host:

   ```
   # rm /usr/adm/atria/rgy/rgy_svr.conf
   ```

3. **Switch registry server host assignments**—On all ClearCase hosts, replace the *rgy_hosts.conf* file, so that it contains the name of the new registry server host:

   ```
   # echo "ccsvr04" > /usr/adm/atria/rgy/rgy_hosts.conf
   ```

4. **Stop and restart ClearCase processing on all hosts**—Execute the startup/shutdown script (for example, */etc/rc.atria*) with a `stop` argument; then execute it again, with a `start` argument.

Renaming the Registry Server Host

ClearCase client hosts cache the name of the registry server host. To rename the network's registry server host:

1. **Shut down ClearCase processing on all client hosts**—Use the architecture-specific command (see the *init_ccase* manual page):

   `# /etc/rc.atria stop` *(shutdown command varies with architecture)*

2. **Switch registry server host assignments**—Do this on *all* ClearCase hosts, as described in Step #3 in the preceding section.
3. **Rename the registry server host**—Make the change using the vendor-supplied procedure or tool.
4. **Restart ClearCase processing on all hosts**—Use the architecture-specific command.

Renaming a ClearCase Host

You must make registry-level adjustments when you rename a host that holds the physical storage for one or more VOBs or views. From the registry's perspective, the rename-host procedure is quite similar to the move-storage-directory procedures described in Chapters 11 and 12:

1. **Make sure that the storage directories are not being used**—If the host is home to one or more VOBs, make sure that all clients unmount those VOBs (*cleartool umount*). If the host is home to one or more views, terminate all processes using the views. (Watch out for shells that are not set to the view, but have it as the *working directory view*)
2. **Shut down ClearCase processing on the host**—Use the architecture-specific command (see the *init_ccase* manual page):

   `# /etc/rc.atria stop` *(shutdown command varies with architecture)*

3. **Rename the host**—Make the change using the vendor-supplied procedure or tool. In most cases, this involves a restart of the operating system, which will also restart ClearCase processing. In any event, make sure that ClearCase processing is restarted on the host.
4. **Adjust the registry entries for each storage director**—Use `register -replace` to update object registry entries; use `mktag -replace` to update tag entries.

Index

Symbols

-hosts map
  remote storage access use, 35
  See Also hosts

A

access
  access-layering
    non-ClearCase access use, 76
  ClearCase data
    across the network (chapter), 23
    importance of user base consistency, 5
    procedures used by processes for, 48
  domains
    network region use as (footnote), 30, 168
  locating
    directories and links, 150
  non-ClearCase
    setting up export views for, 74
  paths
    information, registry server host as repository for, 2
    storage directories and, 23
    user-level, registered in tag registry, 6
  See Also licenses
  See Also users
  source pools
    commands and operations for, 13
  views
    deactivating (chapter), 141
    network-wide facilities, 27
    tag-related problems, resolving, 169
  VOB
    deactivating (chapter), 141
    network-wide facilities, 27
    VOB-tag use for, 12
  VOBs
    concurrent, managed by lockmgr(1MA) process, 10
    tag-related problems, resolving, 169
access control
  as VOB administration task
    characteristics, 4
  ClearCase data
    controlled through user and group IDs, 5
  ClearCase level, 52
  locks
    on type objects, 54
    on VOB objects, 53, 141
    preventing accidental deletion of lock manager socket, 82
  See Also administrator
  settings
    ClearCase data structures, 45
    initialization for VOB objects, 47
  umask(1) setting effect on VOB and view creation, 42
  views
    owner and group roles, 39
  VOBs
    adjusting permissions, procedure description, 45
    non-group members, 63
    owner and group roles, 39

administrator
  network-wide
    importance for smooth ClearCase administration, 39
  See Also access control
  See Also maintenance
  See Also storage
  view of the ClearCase network (chapter), 1
  VOB administration tasks (list), 3
albd_server (Atria Location Broker Daemon) process
  as ClearCase server process, 3, 8
  as license server process on license server host, 2
  as registry server process on registry server host, 2
  See Also servers
  started by ClearCase startup script, 10
aliases
  multiple
    as impediment to global naming, 29
  See Also access
  See Also pathnames
architectures
  multiple
    as impediment to global naming, 29
Atria Location Broker Daemon process
  See albd_server (Atria Location Broker Daemon) process
attributes
  See Also VOBs
  stored in VOB database, 11
automount(1M) utility
  automatic file-system mounting performed by, 28
  ClearCase use, 35
  See Also VOBs

B

backup
  as VOB administration task
    characteristics, 4
    distributed VOB issues, 16
  See Also administrator
  See Also maintenance
  tools
    recommendations, 83
  views
    (chapter), 83
  VOBs
    (chapter), 83
    ensuring a consistent, 84
    partial, 85
block buffer caches
  See caches
branch structures
  See Also VOBs
  stored in VOB database, 11
build avoidance
  See configuration lookup
build(s)
  non-ClearCase host, 34

C

c (cleartext) subdirectory
  as VOB storage pool container, 11
  See Also storage
caches
  block buffer
    flushing, 133
    manipulating to improve VOB performance, 132
    statistics, 133
  cleartext storage pools as, 14
  MVFS
    changing to improve performance (SunOS only), 135
    compiling new sizes into (SunOS only), 137
    customizing, procedure (SunOS 4), 138
    customizing, procedure (SunOS 5), 139
    selecting alternative size defaults (SunOS 4 only), 136

view_server process
changing to improve performance, 140
cautions
against moving database directories to another host
view, 19
VOB, 12
links
absolute pathnames in, 170
unexported disk partitions, 170
VOB symbolic, 59
pathnames
absolute, 170
global, 159
remote
source pools, 114, 134, 171
storage pool management, 16
See Also administrator
storage registry administration, 169
tags
renaming and removing, 170
VOB
and view access problems, 159
database synchronization issues, 85
removing elements from storage, 108
central data repository
See Also registries
set of all VOBs use as, 1
checkin
checkin command
source pools accessed by, 13
versions
locating source container, 149
checkout
checkout command
source pools accessed by, 13
ClearCase server processing steps, 8
versions
locating, 148
chpool (cleartool subcommand)
as tool for working with storage pools, 114
changing storage pool element assignments, 17
chpool command
See Also storage pools
cleaning up
See backup
See maintenance
See scrubbing
clearaudit program
DO storage pools used by, 14
ClearCase
client-server architecture
characteristics, 2
data
network-wide access (chapter), 23
structures, 3
user-level access (chapter), 37
hosts
changing (chapter), 173
network
administrator's view (chapter), 1
servers
characteristics, 8
user base, 5
clearcvt_unix command, 69
clearmake program
DO storage pools used by, 14
cleartext
data containers
contained in cleartext storage pools, 14
locating, 148
See Also files
storage pools
backup considerations, 86
maintenance procedures, 96
usage patterns that impact location, 114
cleartext storage pool, 114
cleartool subcommands
chpool, 17, 114
lspool, 17, 114
lsview -long, 159
lsvob -long, 159
mkpool, 15, 17, 114

mktag, 161
mkview, 160
mkvob, 15, 17, 160
protect -chmod, 46
protectvob, 41
rmpool, 17
rmtag, 161
rnpool, 17
storage pool handling (list), 17
client
hosts
characteristics, 2
improving performance, 133
MVFS required on all, 2
process
characteristics, 8
removing from VOB host to improve performance, 131
programs
database requests handled by db_server process, 12
database requests, server process that handles, 8
See Also views
See Also VOBs
client-server architecture
ClearCase
characteristics, 2
processing characteristics, 8
See Also client
See Also servers
co-locating
VOBs and their export views, 75
compressed files
cleartext data containers used to hold, 14
See Also files
configuration lookup
not available on a non-ClearCase host, 35
performance
enhanced by view-private caching of config recs, 20
configuration records (CRs)
DO
cached in view-private storage, 20
managing the storage of, 98
stored in VOB database, 11
consistency
user base
tools for achieving, 5
user-ID
requirement for, 38
conversion
RCS data
to ClearCase, (scenario), 69
creating
See storage pools
See VOBs (versioned object bases)
crontab(1) scripts
modifications for periodic VOB maintenance procedures, 95
modifying entries in, 80
periodic scrubbing tasks performed by, 4
preventing accidental deletion of data by (chapter), 79
See Also maintenance
cvt_dir directory
master conversion script created in, 70

D

d (DO) subdirectory
as VOB storage pool container, 11
See Also files
data containers
access-control settings, 47
determining the location of (chapter), 145
DO
cached in view-private storage, 20
locating
with mvfsstorage, 147
See Also files

stored in VOB storage pools
types of, 13
stored in VOB storage pools, 11
data repositories
central
set of all VOBs use as, 1
See Also registries
See Also views
See Also VOBs
server hosts as, 3
data structures
network-wide access (chapter), 23
See Also files
See Also views
See Also VOBs
server process management of, 8
user-level access (chapter), 37
data-conversion utilities
clearcvt_unix, 69
db (database) subdirectory
location requirements, 12
See Also directories
See Also files
view storage directory
components and characteristics, 19
view database contained in, 19
VOB storage directory
components and characteristics, 12
VOB database contained in, 11
db_server process
as server that accesses the VOB database, 12
characteristics, 8
See Also processes
deactivating
views and VOBs (chapter), 141
debugging
See cautions
derived object storage pool, 114
describe command
-vob option
listing VOB identity information, 41
as tool for working with storage pools, 114
determining the status of files, 145
directories
locating, 150
lost+found
files within a deleted directory moved to, 110
removing
file elements not affected by, 110
See Also files
See Also views
See Also VOBs
storage
registry tree, moving to another host, 174
server hosts as data repositories for view and VOB, 3
view, access paths and, 23
view, components, 19
view, creating, 74
view, determining a location for, 73
view, server process that handles, 8
view, views implemented as, 4
VOB, access paths and, 23
VOB, cautions against moving, 19
VOB, components, 11
VOB, creating, 62
VOB, .identity subdirectory characteristics as VOB component, 12
super-root
viewroot directory as (figure), 80
/tmp_mnt
automount use, 35
tree
designing multiple VOBs as a virtual, 59
UNIX directory trees
overcoming disk partition restrictions with remote storage pools, 16
/usr/adm/atria/log
error log location, 9
/usr/atria/bin
client program directory, 2
/usr/atria/etc
mvfsstorage utility located in, 148
server program directory, 3

/view
view-tag relationship to, 25
viewroot
as super-root (figure), 80
view-tag relationship to, 25
VOB root directory
creating by mkvob, 17
disk space
conserving
by moving VOB storage pools, 113
by removing uninteresting versions, 108
mounting a partition, 43
See Also storage
distributed client-server application
ClearCase as, 8
See Also client
See Also network
See Also servers
DOs (derived objects)
configuration records (CRs), 98
data containers
removing, 98
stored in DO storage pools, 14
pools
maintenance procedures, 96
usage patterns that impact location, 114
reestablishing consistency of VOB database with respect to, 90
scrubbing
adjusting the procedures for, 117
See Also storage
See Also views
sharing
not available on a non-ClearCase host, 35
storage pools
backup considerations, 85
transferring to VOB storage (example), 99

E

elements
cautions about removing from VOB storage, 108
directory
removing, file elements not affected by, 110
file
not affected by directory element removal, 110
restoring
from backup, procedure, 110
that have been mistakenly removed, 108
See Also files
stored in VOB database, 11
type
as determiner for type of data container file storage, 11
error logs (server)
characteristics and location, 9
See Also backups
See Also errorlogs_ccase(4A) manual page
/etc/exports.mvfs file
non-ClearCase host use, 34
See Also files
/etc/fstab.mfs file
See Also files
/etc/init.d/atria file
as startup script name, 10
See Also files
/etc/logingroup file
HP-UX host group assignment file, 38
See Also files
/etc/rc.atria
as startup script name, 10
/etc/rc.atria script
revising to configure larger default MVFS cache sizes, 136
event records
scrubbing, 97
See Also VOBs
stored in VOB database, 11

export
  mechanisms
    accessing ClearCase data from non-ClearCase hosts using UNIX, 3
  See Also exports_ccase manual page
  See Also hosts
  See Also network
  views
    setting up for non-ClearCase access, 74

F

file system
  See files
file system data
  See files, data
files
  compressed
    cleartext data containers used to hold, 14
  data
    accessing, 45
    as MVFS objects, 11
    determining the status of, with describe, 145
    MVFS, determining the location of (chapter), 145
    MVFS, locating, with mvfsstorage, 147
    not stored in VOB database, 11
    stored in VOB storage pools, 11
    view-private, locating, 149
  elements
    cleartext storage pools as location of cleartext data containers for, 14
    DO storage pools as location of shared DOs, 14
    source pool assignments, 17
    source storage pools as location of source data container for, 13
  /etc/exports.mvfs
    non-ClearCase host use, 34
  /etc/fstab.mfs
    no longer required for mounting VOBs, 151
  /etc/init.d/atria
    as startup script name, 10
  /etc/logingroup
    HP-UX host group assignment file, 38
  extended system
    views as, 18
  file descriptor table
    overall, modifying for a VOB host, 61
  MVFS
    determining the location of (chapter), 145
  See Also directories
  See Also storage
  See Also views
  See Also VOBs
  text
    cleartext data containers used to hold, 14
  /usr/adm/atria/config/alternate_hostnames
    multiple network interfaces recorded in, 33
  /usr/adm/atria/config/automount_prefix
    specifying a non-standard mount directory in, 35
  /usr/adm/atria/log/scrubber_log
    analyzing before adjusting scrubbing parameters, 117
  /usr/adm/atria/rgy/rgy_region.conf
    network region assignments recorded in, 33
  /usr/adm/atria/rgy/vob_tag.sec
    as VOB-tag password file, 62
    passwords maintained in, 27
  /usr/atria/config/vob/vob_scrubber_params
    as per-host VOB configuration file, 97
  view_tags
    view tag registries implemented in, 31
flushing
  block buffer caches, 133
  See Also maintenance
  See Also scrubbing

G

global
  naming
    impediments to, 28
  pathname
    tag registry use, 25
  resources
    views and VOBs seen as
  See Also netowork
groups
  access-control issues, 50
  adjusting VOB identity information to reflect the structure of
    (scenarios), 63
  IDs
    consistency among ClearCase hosts, importance for successful use of ClearCase, 5
    user access control maintained through, 5
  list
    access control managed through, 4
  multiple
    support for VOBs and views, 40
  principal
    access control managed through, 4
  See Also access control
  See Also group(4) manual page
  setting up views for
    procedures, 73
  user
    assigning, 37
  usernames and
    characteristics, 37
growth
  managing
    as VOB administration task, characteristics, 4
  See Also maintenance

H

hosts
  -hosts map
    remote storage access use, 35
  as ClearCase component, 1
  cautions against moving
    view database directories to another, 19
    VOB database directories to another, 12
  ClearCase
    changing (chapter), 173
    user-ID consistency importance for successful use of ClearCase, 5
  client
    characteristics, 2
    improving performance, 133
    MVFS required on all, 2
  host-local pathname
    local VOB and view location recorded by, 24
  HP-UX
    group assignment file, 38
    revising crontab entries, 81
  IRIX
    revising crontab entries, 81
  license server
    characteristics, 2
  non-ClearCase
    ClearCase data structures access from, 34
    creating VOB storage pools on, 113
  OSF/1
    revising crontab entries, 81
  registry server
    characteristics, 2
    registering a new VOB or view, 157
    renaming, 175
  release
    network-wide, characteristics, 2
    renaming, 173
  restricting exports to particular, 78
  See Also network

server
characteristics, 3
types of registries maintained on, 6
SunOS
revising crontab entries, 81
types of
characteristics, 2
VOB
improving performance, 131
modifying for ClearCase access, 61
selecting, 57
HP-UX hosts
group assignment file, 38
revising crontab entries, 81
See Also hosts

I

.identity subdirectory
group list for VOBs and views implemented by, 40
See Also directories
view storage directory, 19
incremental adoption
implications for accessing ClearCase data from non-ClearCase hosts, 34
See Also hosts
infinite looping
preventing, 79
init(1M) program
ClearCase startup script executed by, 10
install_release script
crontab modification by, 80
interfaces
multiple network
as impediment to global naming, 28
See Also network, regions
IRIX host
revising crontab entries, 81
See Also hosts

L

LAN (local area network)
See network
license(s)
active user licensing scheme, 55
ClearCase access control use of (footnote), 5
concurrent users limited by, 1
database file
See Also license.db manual page
floating architecture, 55
See Also access control
See Also hosts
See Also users
server hosts
characteristics, 2
setting up, See Also ClearCase Notebook
server process
See albd_server
links
cautions
against links to unexported disk partitions, 170
against using absolute pathnames in, 170
locating, 150
See Also access
See Also pathnames
symbolic
accessing view-private storage through, 20
remote view and VOB storage accessed through, 23
UNIX, accessing remote VOB storage pools with, 15
VOB directory design use, 59
local
host location
VOBs and views, object registries used to maintain, 24
scripts
caution about maintenance use, 99

location
  logical
    view, 23
    VOB, 23
  physical
    view, 23
    VOB, 23
  See Also access
  See Also hosts
lockmgr(1MA) process
  concurrent VOB access managed by, 10
  preventing accidental deletion of socket
      created by, 82
  See Also processes
locks
  See access control
lost+found directory
  files within a deleted directory moved to, 110
  See Also directories
lspool (cleartool subcommand)
  as tool for working with storage pools, 114
  listing storage pool information, 17
lstag command
  locating views, 147
lsview (cleartool subcommand)
  -long option, as registry tool, 160
lsvob (cleartool subcommand)
  -long option, as registry tool, 160

M

maintenance
  as VOB administration task
    characteristics, 4
  backup
    VOBs and views (chapter), 83
  cleaning up view-private storage
    manually, procedures, 127
    manually, 98
  periodic
    as VOB administration task, characteristics, 4
  See Also administrator
  See Also performance
  user-supplied procedures, 99
  views
    occasional procedures (chapter), 121
    periodic procedures (chapter), 95
  VOBs
    occasional procedures (chapter), 101
    periodic procedures (chapter), 95
memory
  impact on client host performance, 133
  See Also maintenance
  See Also performance
  VOB needs, 57
meta-data
  See Also VOBs
  stored in VOB database, 11
mkpool (cleartool subcommand)
  as tool for working with storage pools, 114
  creating storage pools, 17
  functions performed by, 15
mktag (cleartool subcommand)
  tag registry entries created by, 161
mkview (cleartool subcommand)
  tag and object registry entries created by, 160
mkvob (cleartool subcommand)
  subdirectories created by, 15
  tag and object registry entries created by, 160
  VOB root directory created by, 17
mntrpc_server process
  public VOBs mounted by, 27
  See Also processes
modload(8) utility
  changing dynamic loading of (SunOS 4 only), 153
mount command
  locating VOBs, 147

moving
views
different architecture, procedure, 123
same architecture, procedure, 121
view-private storage, procedure, 126
VOBs
database directory, cautions against, 12
different architecture, procedure, 104
same architecture, procedure, 101
multihop export configurations
limitations and methods for avoiding, 76
See Also hosts
See Also network
MVFS
default configuration, 135
parameters and their values (table), 137
MVFS (multiversion file system)
cache
changing to improve performance (SunOS only), 135
compiling new sizes into (SunOS only), 137
customizing, procedure (SunOS 4), 138
customizing, procedure (SunOS 5), 139
parameters and their values (table), 137
selecting alternative size defaults (SunOS 4 only), 136
changing dynamic loading of (SunOS 4 only), 153
files
determining the location of (chapter), 145
objects, 11
reconfiguring
to improve client host performance (SunOS only), 135
required on all client hosts, 2
See Also directories
See Also files
See Also views
See Also VOBs
mvfsstorage utility
locating MVFS files, 148

N

network
accessing data across (chapter), 23
ClearCase
administrator's view (chapter), 1
overview of components, 1
interfaces
recording, 33
VOB global pathname creation for, 66
multiple-region
registries in, 30
network-wide release host
characteristics, 2
partitioning
reasons for, 163
regions
adding (chapter), 157
adding, procedure, 165
adding, 162
as solution to global naming problems, 28
characteristics and ClearCase data access use, 6
creating, to ensure valid global pathnames, 66
establishing, 33
global pathnames relationship to, 159
inapplicable to network-wide pathnames for remote storage pools, 17
moving hosts to, 168
multiple, handling recommendations, 171
remote storage pools not able to use, 113
removing, 169
tag registry relationship to, 159
See Also registries
NIS
group map
as tools for maintaining user base consistency, 5
passwd map
as tools for maintaining user base consistency, 5

non-ClearCase
  access
    importance of placing VOB and its export view on the same host, 132
    setting up export views for, 74
  hosts
    characteristics, 3
    ClearCase data structures, access from, 34
    ClearCase data structures, restrictions on use, 34
    creating VOB storage pools on, 113
  processes
    removing from VOB host to improve performance, 131
  See Also access
  See Also hosts
  See Also network, regions

O

object registries
  as component of ClearCase storage registry, 158
  characteristics and relation to tag registries, 6
  local VOBs and views location maintained in, 24
  See Also registries
  tag registry relationship to (figure), 26
  tag registry relationship to, 159
OSF/1 host
  revising crontab entries, 81
  See Also hosts

P

password facility
  public VOB access controlled by, 27
  See Also access control
pathnames
  absolute
    cautions against using in links, 170
  consistent
    network regions as tool for, 29
  full
    determining, with ls and pwd, 146
  global
    obtaining information about with lsvob -long and lsview -long, 160
    relationship to network regions, 159
    tag registries used to record, 160
  host-local
    object registries used to record VOB and view, 158
  recursive traversal, 79
  remote location issues, 17
  remote storage pools
    global requirements, 113
  See Also access
  See Also files
  See Also users
  symbolic link
    requirements, 59
    See Also pathnames_ccase manual page
performance
  client
    improving, 133
  See Also maintenance
  SunOS
    changing the MVFS configuration, 135
  tuning (chapter), 131
  views
    reconfiguring to improve, 140
  VOB
    improving, 131
permissions
  ClearCase level control of, 52
  controlled by user and group IDs, 5
  See Also access control
  See Also ct_permissions(5A) manual page
processes
  albd_server (Atria Location Broker Daemon) process, 2, 3, 8, 10

client
characteristics, 8
removing from VOB host to improve performance, 131
db_server process, 8, 12
lockmgr(1MA) process, 10, 82
mntrpc_server process, 27
non-ClearCase
removing from VOB host to improve performance, 131
overall process table
modifying for a VOB host, 61
read access, 51
removing extraneous
to improve VOB performance, 131
servers
characteristics, 8
storage registries used by, 24
serverss
management ofdata structures, 8
view_server process
as ClearCase server program, 3
characteristics, 8
database requests, handled by vobrpc_server process, 12
view_server process, 140
vob_server process
characteristics, 15
vobrpc_server process, 12
write access, 51
product releases
See Also release
VOB use as, 60
protect -chmod (cleartool subcommand)
differences between chmod(1) command and, 46
protectvob (cleartool subcommand)
modifying VOB identity information, 41
public
See Also access
VOBs
activation of, 27
pwv command
locating views, 147

R

regions
See network, regions
registries
ClearCase
managing (chapter), 157
types and characteristics, 6
cleartool commands for working with, 159
in multiple-region networks, 30
managing
as VOB administration task, 3
network-wide
administrative benefits, 6
object
characteristics and relation to tag registries, 6
local VOBs and views location maintained in, 24
tag registry relationship to (figure), 26
tag registry relationship to, 159
See Also hosts
See Also network, regions
server host
characteristics, 2
ClearCase storage registry located on, 157
registering a new VOB or view, 158
renaming, 175
types of registries maintained on, 6
VOB-tag password file located on, 62
storage
administration cautions, 169
administrative benefits, 6
characteristics and use, 24
moving to another host, 174
network-wide, VOBs listed in, 3
See Also registry_ccase manual page
VOB mounting information stored in, 151
tag
characteristics and relation to object registries, 6, 158
characteristics and use, 25
network regions and (figure), 32
object registry relationship to (figure), 26
object registry relationship to, 159

release
  area
    changing location of, 173
    characteristics, 2
  host
    renaming, 173
  See Also hosts
  VOB
    planning for, 60
Release 1 style
  mounting VOBs in, 151
release host
  network-wide
    characteristics, 2
remote storage pools
  choosing a host for, 113
removing
  elements
    cautions, 108
  network regions, 169
  processes
    to improve VOB performance, 131
  See Also maintenance
  See Also scrubbing
  source data
    cautions, 108
  views
    permanently, with rmview, 144
  VOBs
    permanently, with rmvob, 144
restoring elements
  from backup
    procedure, 110
  See Also maintenance
  that have been mistakenly removed, 108
rlogin(1) program
  modifying non-ClearCase host access
      restrictions with, 34
rmelem program
  cautions about using, 108
rmpool (cleartool subcommand)
  deleting storage pools, 17
rmtag (cleartool subcommand)
  tag registry entries removed by, 161
rmver program
  removing uninteresting versions with, 108
rnpool (cleartool subcommand)
  renaming storage pools, 17
root
  directory
    VOB, creating by mkvob, 17
  See Also directories
  user
    crontab(1) scripts set up for, 4
RPC call
  ClearCase server process actions triggered by, 8

S

s (source) subdirectory
  as VOB storage pool container, 11
  See Also directories
  view storage directory
    moving, 21
    view data storage area contained in, 5
  view storage directory, 19
schema
  embedded VOB database
    updating by reformatting the VOB, 4
scripts
  automating type object copying with, 68
  master conversion script, 70
  RCS data conversion
    (scenario), 69
scrubbing
  as VOB administration task
    characteristics, 4

impact on cache hits, 14
parameters
storage pool scrubbing controlled by, 97
scrubber(1MA) utility
deleting data containers from cleartext and DO storage pools, 18
storage pools scrubbed by, 97
See Also maintenance
See Also removing
storage pools
adjusting the default procedures, 117
term definition, 95
view-private storage, 99
VOB
databases, logical vs. physical implications, 97
databases, 97
storage pools, 96
servers
ClearCase
characteristics, 8
error logs, 9
hosts
characteristics, 3
types of registries maintained on, 6
processes
characteristics, 8
host-local pathnames used by, 158
storage registries used by, 24
programs
location, 3
registry server host
characteristics, 2
See Also client
See Also hosts
that access the VOB database
names and functions, 12
setview command
view activation by, 19
shared
DO
stored in DO storage pools, 14
See Also access control
See Also views
views
access-control issues, 49
central administration required by, 1
shutdown script
actions performed by, 10
adjusting (chapter), 151
names
architecture-specific (list), 151
names, architecture-specific (list), 151
See Also init_ccase manual page
site_prep program
establishing network regions, 33
See Also network, regions
size
view_server process cache
default, 140
source
data
cautions about removing from VOB storage, 108
containers, stored in VOB source storage pools, 13
pools
file element assignments, 17
maintenance procedures, 96
recommendation against remote location of, 134
usage patterns that impact location, 114
See Also storage
See Also views
See Also VOBs
speed (CPU)
See Also performance
VOB needs, 58
startup script
actions performed by, 10
adjusting (chapter), 151
names
architecture-specific (list), 151
See Also init_ccase manual page

startview command
  view activation by, 19
stat(2) records
  ClearCase data access information
    maintained in, 45
statistics
  block buffer caches, 133
storage
  data
    as ClearCase component, 1
    ClearCase, characteristics, 3
    data structures that implement VOBs and views
      (chapter), 11
    views, characteristics, 5
  directories
    See storage directories
  disk
    VOB needs, 58
  physical locations
    registered in object registry, 6
  pools
    See storage pools
  registries
    See storage registries
  remote data
    implementation of, 23
  See Also administrator
  See Also files
  See Also views
  See Also VOBs
  short-term
    provided by views, 4
storage directories
  See Also storage
  view
    access paths and, 23
    components, 19
    creating, 74
    determining a location for, 73
    physical location, 23
    server hosts as data repositories for, 3
    server process that handles, 8
    views implemented as, 4
  VOB
    access paths and, 23
    cautions against moving, 19
    creating, 62
    .identity subdirectory characteristics as VOB
      component, 12
    physical location, 23
    server hosts as data repositories for, 3
storage pools
  access-control settings, 47
  backup, 114
  changing element assignments, 15
  cleartext, 114
    as caches for text and compressed files, 14
  commands for working with (list), 17
  creating, 15
  default
    characteristics, 15
    names for, 149
  derived object, 114
  DO
    characteristics and components, 14
    subdirectory name, 15
  local
    characteristics, 15
    creating, 15
  moving
    to another disk, example, 116
  remote
    cautions, 16
    characteristics, 15
    creating, during VOB creation, 67
    creating, example, 115
    creating, to conserve VOB disk space, 113
    creating, to improve client host
      performance, 134
    creating, 15
    locating files in, 149
    VOB, when not to use, 171

scrubbing
adjusting the default procedures, 117
See Also storage
server process that handles, 8
source
characteristics and components, 13
commands and operations that access, 13
subdirectory name, 15
subdirectories (list), 15
tools for working with, 114
views
See views, view-private storage
storage registries
administration cautions, 169
administrative benefits, 6
characteristics and use, 24
components and structure, 157
moving to another host, 174
network-wide
VOBs listed in, 3
See Also administrator
See Also registries
See Also registry_ccase manual page
See Also storage
See Also views
See Also VOBs
VOB mounting information stored in, 151
subdirectories
See Also directories
See Also files
storage pools (list), 15
SunOS
host
revising crontab entries, 81
performance
changing the MVFS configuration, 135
See Also hosts
super-root directory
See Also directories
viewroot directory as (figure), 80
symbolic links
accessing view-private storage through, 20
remote view and VOB storage
accessed through, 23
See Also access
See Also links
See Also pathnames
UNIX
accessing remote VOB storage pools with, 15
VOB directory design use, 59
system resources
increasing
to improve client host performance, 133
See Also administrator

T

tag registries
characteristics
and relation to object registries, 6
and use, 25
implementation
in a multiple-region network, 31
network regions and (figure), 32
object registry relationship to
(figure), 26
object registry relationship to, 159
See Also registries
See Also view_tags
See Also vob_tags
tags
removing
cautions against explicit, 170
renaming
cautions against, 170
text files
cleartext data containers used to hold, 14
See Also files
/tmp_mnt directory

automount use, 35
See Also directories
/tmp/.A/almd socket
preventing accidental deletion of, 82
See Also access control
type manager program
text_file type
characteristics, 13
type objects
coordinating
for multiple VOBs, 68
copying, 68
locking, 54
See Also ClearCase, data
stored in VOB database, 11

U

umask(1) setting
effect on VOB and view creation, 42
See Also access control
setting for a shared view, 73
umask(1) value
affect on view access, 73
UNIX directory trees
overcoming disk partition restrictions
with remote storage pools, 16
VOBs implemented as, 11
unshared DOs
as major storage drain on view-private storage, 20
See Also DOs
users
access control
maintained through, 5
concurrent
limited by license, 1
data structure access by (chapter), 37
individual
setting up views for, 71
multiple
setting up views for, procedures, 73
See Also access control
See Also pathnames
user base
architecture and access structure (chapter), 37
as ClearCase component, 1
user-IDs
consistency, importance for successful use of ClearCase, 5
consistency, requirement for, 38
usernames
groups and, characteristics, 37
/usr/adm/atria/config/alternate_hostnames file
multiple network interfaces recorded in, 33
See Also files
See Also hosts
/usr/adm/atria/config/automount_prefix file
See Also files
specifying a non-standard mount directory in, 35
/usr/adm/atria/config/license.db file
See Also files
/usr/adm/atria/license_host file
See Also files
See Also hosts
/usr/adm/atria/license.db file
See Also files
/usr/adm/atria/log directory
error log location, 9
/usr/adm/atria/log/scrubber_log file
analyzing before adjusting scrubbing parameters, 117
See Also files
/usr/adm/atria/rgy directory
ClearCase storage registry contained in, 157
See Also directories
/usr/adm/atria/rgy/rgy_region.conf file
network region assignments recorded in, 33
See Also files
/usr/adm/atria/rgy/vob_tag.sec file
as VOB-tag password file, 62

passwords maintained in, 27
See Also files
/usr/atria/bin directory
as client program directory, 2
See Also directories
/usr/atria/config/cron/ccase_local.day script
as daily VOB maintenance script, 99
See Also maintenance
/usr/atria/config/cron/ccase_local.wk script
as weekly VOB maintenance script, 99
See Also maintenance
/usr/atria/config/vob/vob_scrubber_params file
as per-host VOB configuration file, 97
See Also files
/usr/atria/etc directory
as server program directory, 3
mvfsstorage utility located in, 148
See Also directories
/usr/atria/examples/rmver_all script
removing uninteresting versions with, 108
utility commands
that work with storage pools (list), 18

V

versions
labels
stored in VOB database, 11
removing
unneeded, 108
See Also ClearCase, data
stored in VOB database, 11
uninteresting
term definition, 108
version-control information
stored in VOB database, 11
view access
affect of umask(1) value, 73
view storage area
nonlocal data storage, 72
view_scrubber command
moving data containers from view-private storage to VOB DO storage pool, 18
view_scrubber utility
manually removing redundant DO data containers, 99
views
accessing
access path information contained on registry server host, 2
access-control settings, initializing, 47
access-control settings, 45
ClearCase data access requirement of, 48
making inaccessible (chapter), 141
network-wide facilities, 27
owners and groups, 39
tag-related problems, resolving, 169
with pwv and lstag, 147
activating, 27
as ClearCase component, 1
backing up, 91
characteristics and components, 18
data structures that implement (chapter), 11
database
characteristics and components, 19
characteristics and use, 19
characteristics, 5
storage requirements, 72
directories
/view, view-tag relationship to, 25
viewroot, as super-root (figure), 80
viewroot, view-tag relationship to, 25
distributed
implementation, 23
export
setting up, for non-ClearCase access, 74
identity
directory, characteristics and components, 19
information, modifying, 41
impact of largeinit MVFS configuration on, 135

location
  accessing with pwv and lstag, 147
  architectural constraints that impact the, 71
maintenance
  occasional procedures (chapter), 121
  periodic procedures (chapter), 95
moving
  different architecture, procedure, 123
  same architecture, procedure, 121
multiple
  export issues, 75
  group support for, 40
network-wide access (chapter), 23
private storage area
  characteristics and components, 19
  characteristics and use, 20
  locating files, 149
  moving, procedure, 126
  remote location access, 20
  scrubbing, 99
  storage requirements, 72
reconfiguring
  to improve performance of, 140
reestablishing consistency of derived object state, 90
registering
  on the registry server host, 157
registries
  object, local view location maintained in, 24
registries, 6
removing
  from VOB host to improve performance, 132
  permanently, with rmview, 144
restoring
  from backup, 92
  to service, 144
See Also ClearCase, data
See Also storage
See Also view(4A) manual page
See Also VOBs
setting up (chapter), 71
shared
  central administration required by, 1
storage
  maintenance considerations and procedures, 98
  registries, characteristics and use, 24
  remote storage pool facility compared with VOB remote storage pools, 20
  requirements, 72
  short-term, for data created during the development process, 4
storage registry
  components and structure, 157
taking out of service, 144
view access by, 19
.view file
  unaffected by moving a view, 123
view_server process
  as ClearCase server program, 3
  cache, changing to improve performance, 140
  characteristics, 8
  database requests, handled by vobrpc_server process, 12
  database requests, server process that handles, 8
  location of, 71
view_tags
  characteristics and use, 25
  choosing for a shared view, 73
  ClearCase data accessed through, 3
  file, view tag registries implemented in, 31
  logical view location specified by, 23
  registry, multiple-region network implementation, 30
VOB
  administrator point-of-view contrasted with developer point-of-view, 4
  coordination importance, 68
  interactions, access control, 46
virtual workspace
  See views
VOBs (versioned object bases)

access
  access control, 53
  access path information contained on registry server host, 2
  concurrent, managed by lockmgr(1MA) process, 10
  ensuring, strategies for, 65
  initializing access-control settings, 47
  making inaccessible (chapter), 141
  network-wide facilities, 27
  tag-related problems, resolving, 169
  umask(1) setting effect on VOB and view creation, 42
accessing
  See VOB-tags
activating, 27
administration tasks (list), 3
as central data repository, 1
as ClearCase component, 1
backing up
  components for partial backups (table), 85
  remote storage pools, procedures, 86
backing up, 83
contents
  See Also vob(4A) manual page
coordinating, 68
creating
  data structures that implement (chapter), 11
  procedure description, 43
creating, 62
databases
  characteristics, 12
  kinds of data stored in, 11
  scrubbing, 97
disk
  space, conserving by moving storage pools to remote locations, 113
  storage needs, VOB, 58
distributed
  backup issues, 16
  implementation, 23
hosts
  capacity of, 58
  data allocation tradeoffs, 58
  improving performance, 131
  kernel resources, modifying, 61
  modifying for ClearCase access, 61
  resources required for, 61
  selecting, criteria for, 57
  selecting, 57
identity information
  adjusting, (scenarios), 63
  adjusting, 63
  listing with describe -vob, 41
  modifying, with protectvob, 41
locating, 84
locking, 141
maintenance
  growth management (figure), 96
  occasional procedures (chapter), 101
  periodic procedures (chapter), 95
  storage tradeoffs, 95
memory needs, 57
mounting
  in Release 1 style, 151
moving
  different architecture, procedure, 104
  same architecture, procedure, 101
multiple
  exporting, 75
  group support for, 40
  linking into a single directory tree, 59
network-wide access (chapter), 23
object registries
  as component of ClearCase storage registry, 158
  local VOB location maintained in, 24
owners and groups
  information located in .identity subdirectory of VOB storage directory, 11
owners and groups, 39
performance
  improving, 131

permanent data repository distributed among, 3
planning for, 58
populating with data
strategies for, 68
private
activation requirements, 27
public
implications of, 63
reformatting
as VOB administration task, characteristics, 4
registries, 6
release
planning for, 60
remote storage pool facility
compared with view remote storage pool, 20
removing
permanently, with rmvob, 144
unneeded versions from, 108
restoring
from backup, procedures, 87
to service, procedure, 143
root directory
created by mkvob command, 17
See Also views
See Also vob_scrubber manual page
setting up (chapter), 57
speed (CPU) needs, 58
storage directories
access paths and, 23
cautions against moving, 19
components, 11
creating, 62
placing in root partition, 43
See Also VOB database
See Also VOB identity directory
See Also VOB storage pools
server hosts as data repositories for, 3
VOB implemented as, 3
storage pools
characteristics, 13
components, 12
database relationships, 13
moving data containers from view-private storage to, 18
scrubbing, 96
source, characteristics, 13
subdirectories (list), 15
vob_server process characteristics, 15
storage registries
characteristics and use, 24
components and structure, 157
storage registry
components and structure, 157
taking out of service, procedure, 141
view interactions
access control, 46
VOB-tags
characteristics and use, 25
ClearCase data accessed through, 3
creating during VOB creation, 62
logical VOB location specified by, 23
multiple-region network, implementation, 30
multiple-region network, recommendations, 171
password file location, 62
vob_scrubber(1MA) utility
event records scrubbed by, 97
vob_server process
characteristics, 8, 15
vob_server program
as ClearCase server program, 3
vobrpc_server process
as server that accesses the VOB database, 12
characteristics, 8

W

warnings
symbolic link pathnames, 59
wink-in
DO storage pools populated through, 14
not available on a non-ClearCase host, 35
See Also DO's
system actions relative to the view-private storage area, 20

Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-1774-020.

Thank you!

Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-965-0964
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California 94043-1389