Table of contents

Class Introduction

- This is the site for the July 2024 Class *ClearCase Administration with MultiSite*

Logistics

- The class runs from 9am to 5pm from July 17 to July 19
- Lunch is from 1pm EDT to 2pm EDT without fail. This gives you a predictable block of time to manage out of class issues like returning calls and scheduling meetings.
- There is a 15min break mid-morning-ish and another mid-afternoon-ish. The actual time will depend on where we are in the flow of the material.
- There will be no new material presented after 4pm unless we are running behind schedule. The last hour will be reserved for additional topics, individual help and to answer questions. Do take advantage of that opportunity.

Introductions

Rod Davison

Rod davison

I have been involved in the technology industry for over 50 years, starting in 1972

- Academia (theoretical mathematics, linguistics, cognitive science, psychology)
- Artificial Intelligence R&D and product development since 1982
- Software development, commercial products and large scale systems (e.g., compiler development and computerizing stock exchanges)
- Data Analytics – Social Research and professional Market Research
- Project Manager and Research Director
- Quality and Testing Professional and Manager
- Business Analysis and forensic investigation
- Consulting and training in advanced technology, management and
- I have worked with government agencies in Canada and the US, defence contractors and aerospace firms, major financial institutions (many years on Wall Street NYC) since 1982.

And now...

Introduce yourself

Please introduce yourself briefly using the following as a guide.

- Name you prefer to be called in class, if different from the one listed on the class list
- Your area of professional focus (developer, SRE, tester, BA, hardware engineer, sysadmin, etc.)
- Any experience you have with ClearCase or other version control systems.
- Any specific objectives for the class or specific takeaways you are looking for
- Just to humanize things, tell us one non-technical thing about yourself, like a hobby or special interest.

Class Materials

This site will be the primary resource for the class; however, there is a GitHub resources repository (https://github.com/ExgnoRepos/2417-CCAdmin-July-17) that contains

1. A pdf generated from this site
2. Additional materials and resources
3. Lab VM assignments

VMS

- You will each be provided with a VM and your own copy of RQM.
- In the first lab, the instructor will walk you through the process of logging into the VM and starting the application
- The VMs will be available continuously until at least 6pm on July 12

Class Protocols

The measure of how effective this class is how much you learn as opposed to the amount of material thrown at you during these four days.

In keeping with that, my classes are run under the following protocols.

- **Questions are ALWAYS appropriate.** Ask questions when they occur to you and are fresh in your mind. Normally, I notice when someone raises their hand, but I don't seem to have notices, then do unmute yourself and get my attention.
- At various points during the class, I will run through the participant list and check to see how you are coming along with the material. This is my measure of how well we are doing as a class, so if you feel lost or confused, this is when you tell me. Almost always, if you are confused, it's because I didn't explain something well enough.
- If you know in advance that if you are going to be late or away for part of the class, let me know in advance, so I don't hold the class up waiting.
- **Have Fun!!!**

Introduction to ClearCase

History of ClearCase

ClearCase Design Objectives

ClearCase is designed around two fundamental principles – first that it should be as easy as possible to use from the user's perspective; and second, that it should perform the saving and retrieving of artifacts as efficiently as possible from the system resource perspective.

User Perspective

The first design goal of ClearCase is that the product should be as invisible as possible to the end user. As long as there are robust work processes in place, then ClearCase should be able to work everywhere in the background without the user ever really having to even be aware that they are using ClearCase. To do this, ClearCase:

1. Makes items under version control look exactly like regular files in the native file system. This means there is no learning curve for working with files in ClearCase; users continue to work with files as they always have.

2. Ensures that make utilities and other tools like awk and grep, and in particular user scripts, run without modification when applied to files under version control. There is no need to "port" or re-write any scripts or tools in development environments to work with files under version control.

3. Automates ClearCase actions by using *"triggers"* so that events in a programmer's IDE or other application can invoke ClearCase actions.

4. Has a Unix like command line set of commands that can be integrated into standard shell scripts.

5. Models its commands after the Unix shell command set so that anyone who is familiar with Unix commands will find the ClearCase command set intuitive.

The last goal may seem a bit odd nowadays, but in 1992 ClearCase was designed to be used by organizations where either Unix or mainframes were the main development platforms for large projects.

System Perspective

A second ClearCase design goal was to make ClearCase as efficient as possible and easy to administer. This means that ClearCase:

1. Decouples how elements under version control are viewed by users from how they are actually stored. The data storage mechanisms used for managing stored are optimized for performance.
2. Decouples the user's perspective of where the VOBs are from their actual physical location so that administrators can restructure their physical infrastructure without the users even being aware of the changes.
3. Allows administrators to implement security, backup and other administrative policies without requiring the user to be aware of them.

Conceptual Architecture

It's important to keep in mind that this architecture was implemented in the late 1980s when there was no public internet and even client server architectures were in their infancy.

The target market for CC was large organizations where large teams of developers needed to collaborate on projects which usually had massive codebases. Many early adopters of CC were large government agencies like the IRS, DoD and organizations that worked on large scale computer engineering projects, like Lockheed-Martin, Boeing and others.

The version control systems before CC were intended for either individual developers or small teams of developers. These systems did not scale well in terms of code size or number of collaborators, and tended to be very basic in terms of functionality and audit features.

CC was originally designed as not just a repository but an entire industrial strength ecosystem to manage version control in large multi-host environments.

Critical to this was the idea of a *CC site*; a network of computers that CC managed to provide the infrastructure support in a multi-host environment.

MVFS Infrastructure

One of the key components of the CC infrastructure is the *Multi-Version File System* or MVFS. This is a synchronous remote procedure call (RPC) network protocol used by the different CC hosts to communicate with each other. It's not strictly a protocol but sits on top of a set of CC network components that integrate with operating system's network and file system layer.

Because MVFS is synchronous, the host, which is making a network request, waits for the remote host (usually a View or VOB server) to complete the requested operation and return a response before it continues its process. Because of this, the client is blocked until the response is received from the remote host.

This approach works well when accessing a file or other version-controlled objects that usually requires an immediate response. For example, when a user requests to read a file managed by ClearCase, the MVFS sends a synchronous RPC to the ClearCase server. The server responds with the file's data, and only then MVFS can return the data to the user. While synchronous RPCs can potentially introduce latency, as it involves waiting for responses, in the case of MVFS, this mechanism ensures the integrity and consistency of the data being accessed or modified.

However, MVFS has not aged well for a number of reasons

1. *Performance Issues:* MVFS, due to its synchronous nature, can suffer performance issues over high-latency networks, such as typical internet connections. Unlike local area networks (LAN), internet connections can frequently have higher latency and less stability, leading to slower response times for read/write operations.

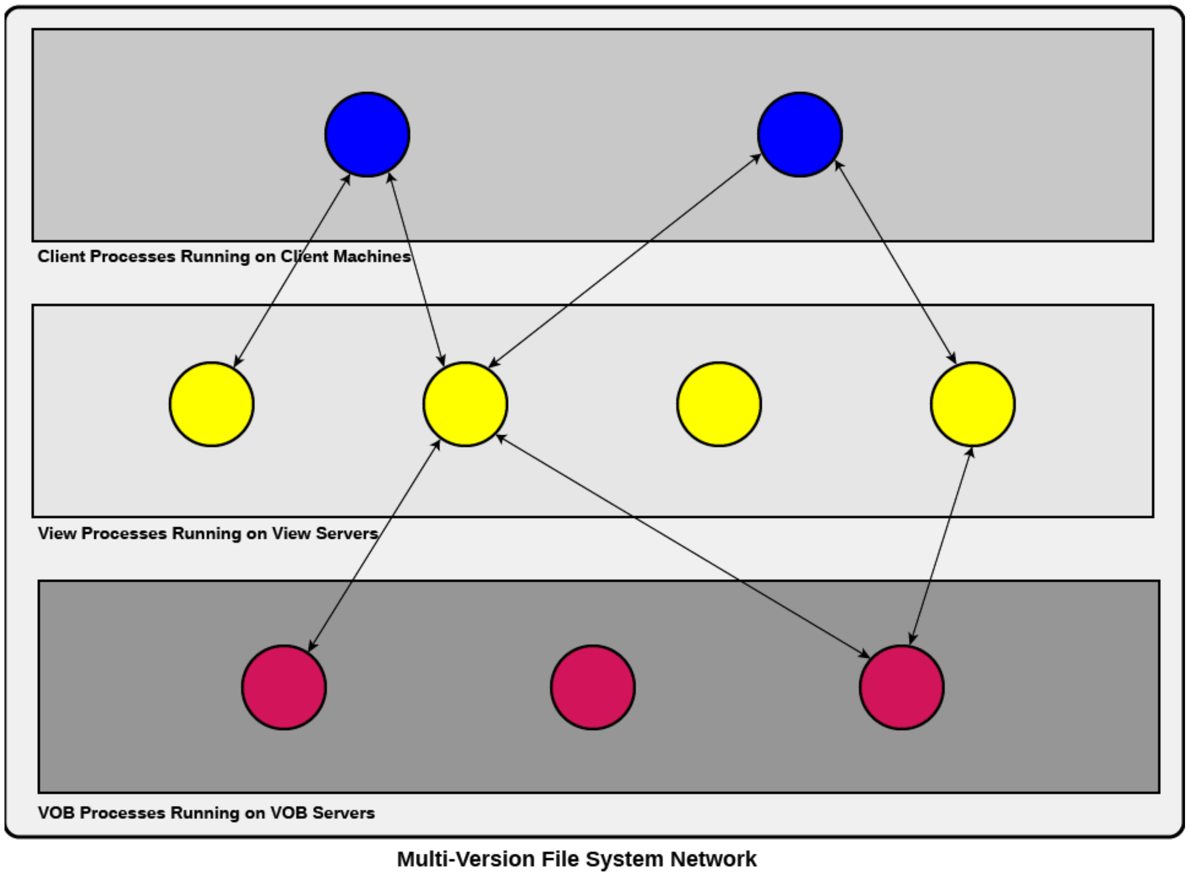
2. *Compatibility Concerns:* Since MVFS is a kernel-level component that needs to be installed on your operating system, sometimes it could have compatibility issues with newer versions of an operating system, especially if the ClearCase version installed is outdated.

3. *Network Security:* Relying on a client-server model through the internet can pose security risks. VPNs or secure network tunnels are required to ensure the secure transmission of data.

4. *Distributed Development Bottlenecks:* In a distributed development environment where team members are geographically dispersed, there may be bottlenecks due to the limitations of MVFS when handling operations over the network.

The Three Tiers

Conceptually, CC implements what came later to be known as a three-tier architecture.


Client Processes Running on Client Machines
View Processes Running on View Servers
VOB Processes Running on VOB Servers
Multi-Version File System Network

Conceptual architecture

Each of these tiers can be thought of one of the specific types of CC processes running in the MVFS environment.

The Client Tier

- This tier is where the client CC process runs on the end user's local machine, which can be a UNIX or Windows host.
- The primary function of the client process is to monitor the file system requests made by the user.
- If the request involves a regular file system object (specifically, not an element under CC version control), the process passes it through to the local OS to be processed.
- If the request involves an element under CC version control, the client process intercepts the call, formulates a request to the appropriate View process and waits for a response.

- Once a response is received, it is displayed exactly in a format indistinguishable from a regular file system operation.
- The client tier also supports the CT shell that allows CC system commands to be executed from the client host.

The View Tier

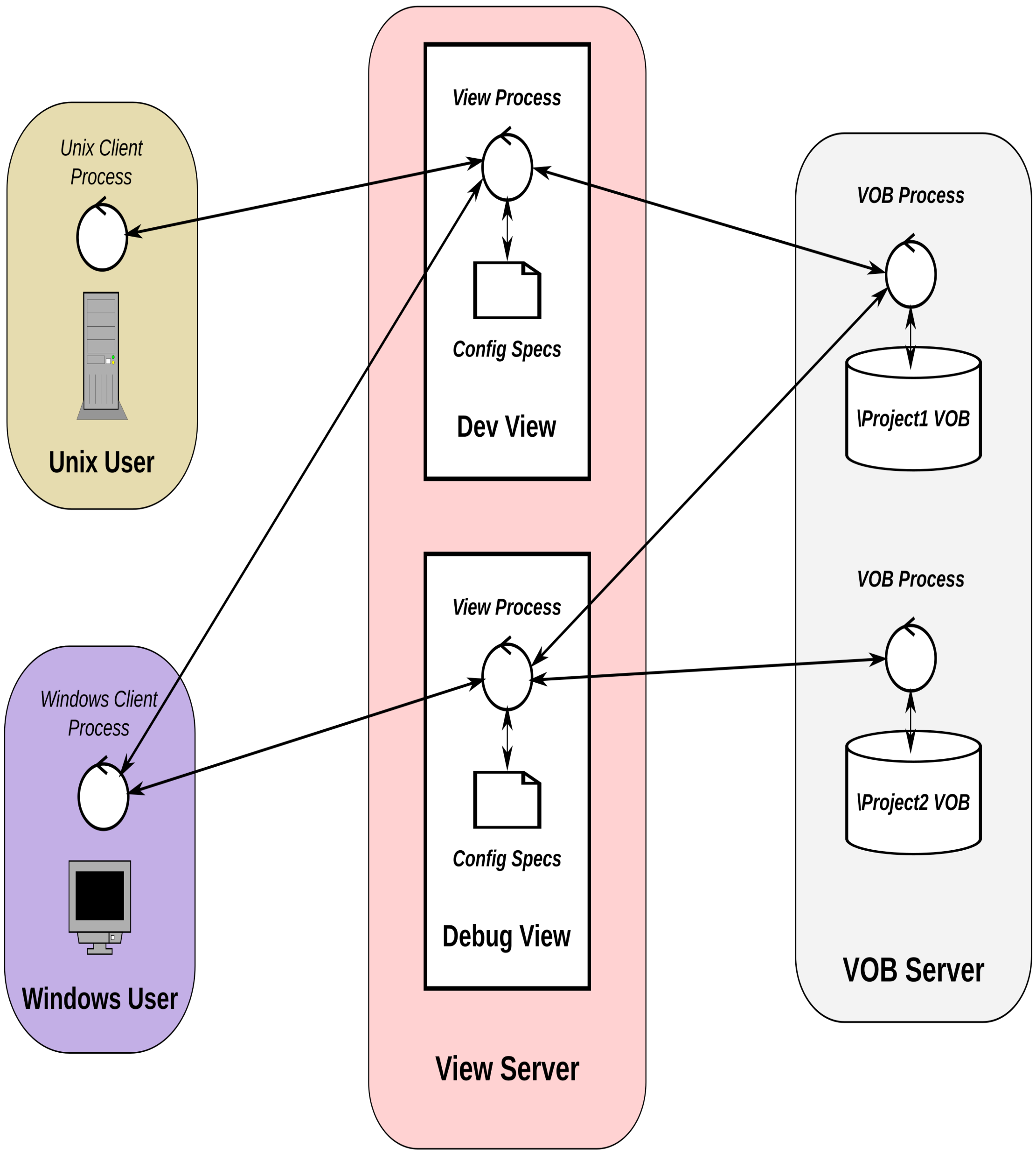
- View processes run in the View tier.
- Any host running a View process is called a View Server, and typically runs a number of View processes.
- A View process receives requests from a client, parses it and then using its configuration specs if necessary, constructs a request for a specific version of a CC element which is sent to the appropriate VOB process.
- The View process requests blocks until a response is received from the VOB process
- Once the reply is received from the VOB process, a reply is formulated and sent to the client process.
- Each View has its own associated View process

The Data Tier

- Any host running a VOB process is a VOB server
- A VOB server receives a request from a View process and performs the appropriate actions (usually some CRUD operation) on the specified element versions and returns the result to the calling View process.
- The VOB process can also respond to direct CC commands for administrative purposes.
- Like Views, each VOB has its own VOB process.

Virtual Architecture

Even though the design of CC predated our standard Internet web-based architecture of URLs, DNS services and IP addresses, the developers created a similar sort of virtual architecture that could be modelled independent of any physical hosts.



Arch

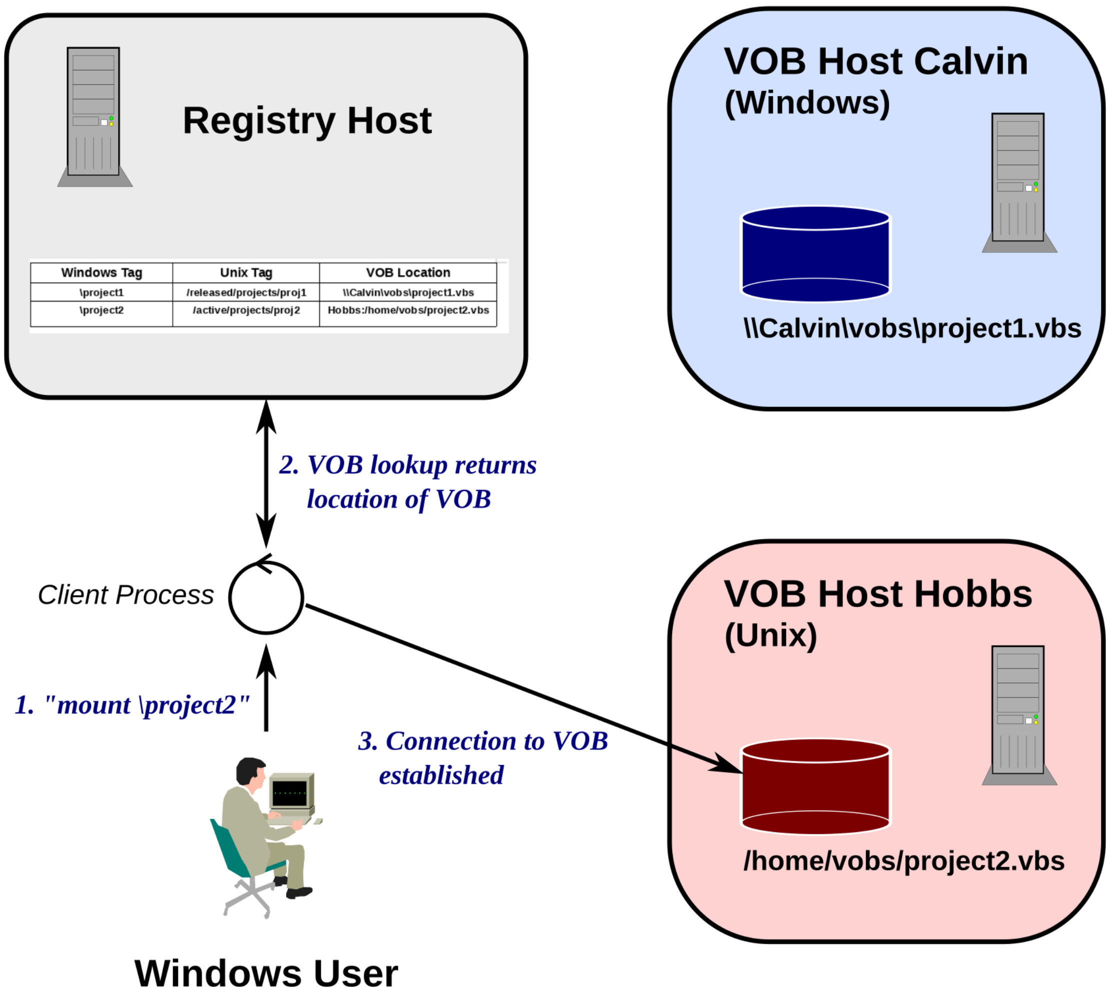
- The VOBs can be thought of as websites. The VOB tag functions like the URL of a website by providing a reference to the VOB without having to know its actual physical location.
- The Views can be thought of as services that can be invoked using their tag without knowing where the View actually is; sort of like invoking a web service over the Internet

- The MVFS and associated server processes provide the network support that we now associate with HTTP and other protocols.

The Registry

The key to getting all these components working together is the Registry Server which functions like a DNS server for websites.

- A DNS server takes a URL and returns the IP address of the host where the site is located.
- Similarly, the Registry server takes a symbolic tab name for a VOB or View and returns the physical location of the VOB
- The host that the Registry server process is running on is called the RRegistry host.
- Note that the VOB tag can look like a Windows network pathname or a UNIX file system path name, which means the Registry may have to associate each VOB with two different tags.



Windows Tag	Unix Tag	VOB Location
\project1	/released/projects/proj1	\\Calvin\vobs\project1.vbs
\project2	/active/projects/proj2	Hobbs:/home/vobs/project2.vbs

Registry

Mounting a VOB

Before a VOB can be used, it has to be mounted which essentially resolves the VOB tag to a specific directory on a VOB server somewhere.

The process CC follows to mount a VOB is:

1. *VOB mount request:* The process begins when a user issues a mount command for a specific VOB on their local machine.
2. *Communication with VOB Server:* The request is sent from the local machine's ClearCase client to the ClearCase VOB server. This request is routed by the Registry server which ensures the mount request is sent to the correct VOB server.

3. *VOB Verification:* The VOB server verifies the existence and access permissions of the requested VOB.
4. *Mounting VOB on MVFS:* If everything is verified, the VOB server provides the necessary information to access the VOB's data. The local ClearCase client then resolves the VOB tag into an MVFS reference
5. _VOB Access: _ Once VOB is mounted, any access to the files in that VOB filter through the MVFS, which communicates with the VOB server in a client-server model via RPCs to retrieve the appropriate version file or to update a file.
6. *File System Mount:* The final stage of the process is to make the mounted VOB look like a mounted filesystem to the client operating system. Since Windows does not support removable file systems, the VOB is presented as the contents of a network drive.

Unmounting a VOB

The reverse process of unmounting a VOB is as follows:

1. To preserve the integrity of the data in a VOB, before any action can be taken to alter any of the VOBs properties, like its physical location, it has to be unmounted.
2. If the VOB is currently in use by any client process, such has being located in the VOB via a View (what this is exactly will be explained later), CC will not allow the VOB to be unmounted. CC will generate a *Cannot unmount VOB currently in use* error.
3. If the VOB is not in use, the MVFS file system references to it are removed which physically disconnects the VOB from the network and any open network connections to the VOB are closed
4. The client OS is advised that the VOB file system has been unmounted.

Activating a View

A client process can only talk to one view at a time. To use a specific view in UNIX, the `cleartool setview <viewtag>` command is used. This is different from the `startview` command that starts the View process associated with the View.

The `lsview` CT command can be used to list all the available views and which one is active. In the example below `myview` is the active view

```
$ cleartool lsview
  * myview             /views/myview.vws
    devview            /views/devview.vws
    qaview             /views/qaview.vws
```

In Windows, views are represented by network drives and activating a view is done by locating to that network drive. This means that all the views are essentially active all the time. Or more realistically, the idea of an active view in Windows is vacuous. The above display in Windows is

```
Y:\>cleartool lsview
* myview
\\COMPUTER000\views\COMPUTER000\Administrator\myview.vws
* devveiw
\\COMPUTER000\views\COMPUTER000\Administrator\devveiw.vws
* qaview
\\COMPUTER000\views\COMPUTER000\Administrator\qaview.vws
```

Storage Locations

- A storage location is a dedicated storage directory on a VOB or View server that has a symbolic name independent of its actual filesystem name.
- Storage locations are given unique IDs, so they can be tracked by the Registry

```
cleartool> lsstgloc -l
Name: vob_storage
  Type: VOB
  Region: COMPUTER000
  Storage Location uuid: 9809ba27.c3914414.b589.07:1a:ff:1f:f5:4d
  Global path: \\COMPUTER000\vobs
  Server host: COMPUTER000
  Server host path: C:\vobs

Name: view_storage
  Type: View
  Region: COMPUTER000
  Storage Location uuid: 0b33ee0d.12d34d4e.8f30.3f:ad:1b:a0:1e:ea
  Global path: \\COMPUTER000\views
  Server host: COMPUTER000
  Server host path: C:\views
```

In IBM Rational ClearCase, a storage location (often abbreviated as `stgloc`) serves a specific and crucial purpose in managing the storage of versioned data.

There are a number of reasons for using storage locations:

1. **Centralized Storage**: Storage locations are centralized repositories that ensure consistency and ease of access across development teams. T
2. **Repository for VOBs**: By using storage locations, ClearCase organizes and manages these VOBs efficiently as opposed to having VOB being created in arbitrary locations in the file system.
3. **Management of View Storage**: Storage locations are also used for storing view storage directories. The storage location keeps a View's private storage, which includes view-private files and metadata specific to that view.
4. **Ease of Administration**: Using storage locations simplifies administrative tasks. Administrators can define storage locations in a manner that supports their organization's data management and backup strategies. It allows for easier maintenance, backup, and replication of versioned data.
5. **Scalability**: Storage locations help in scaling the ClearCase environment. By distributing VOB and View storage across multiple storage locations, organizations can manage large volumes of data more effectively and ensure better performance and reliability.
6. **Network Efficiency**: By placing storage locations strategically within the network, organizations can optimize access speeds and reduce network load. Developers can access the storage location closest to them, improving performance and productivity.
7. **Backup and Recovery**: Storage locations facilitate systematic backup and recovery processes. Administrators can back up entire storage locations to ensure that all versioned data is preserved and can be restored in case of data loss or corruption.

Overall, the benefits of using of a storage location in ClearCase are to provide a structured, efficient, and manageable repository for versioned data, supporting robust configuration management and collaborative software development.

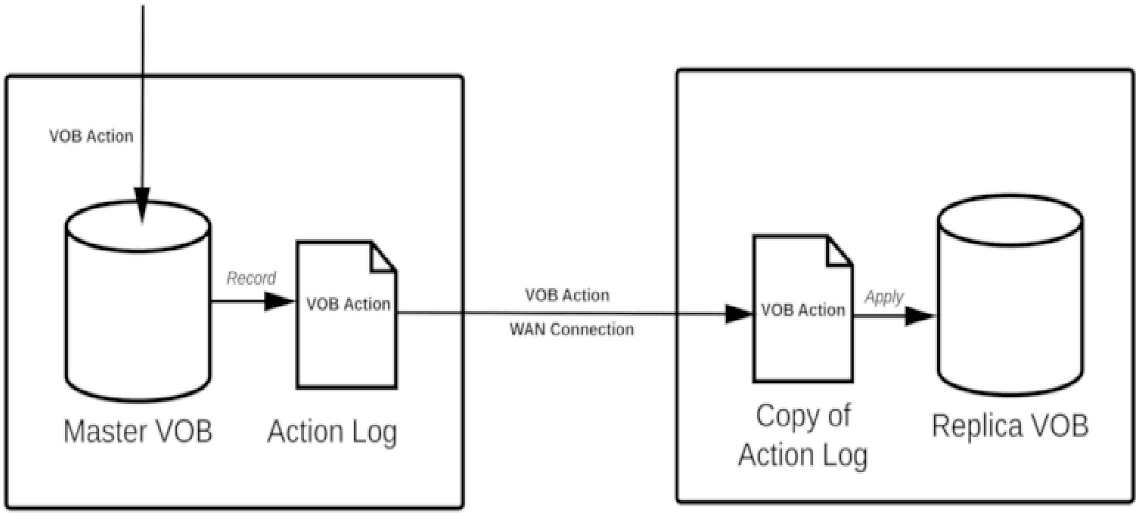
MultiSite

ClearCase was developed in an era where networking was primarily site-specific LANS. For example, all the computers within a university campus might be networked or all the computers in the New York office of a company. However, WANs or wide-area networks were not common. For example, the predecessor to the Internet, ARAPANET, was being used to connect some universities, government agencies and companies, but it was a private club.

Because CC targeted companies that often needed their geographically distributed teams to collaborate, a solution that could piggyback on the WANs of the time was required. This solution was multi-site.

Realtime synchronization was not possible given 1980s-1990s technology, so a common approach was a replica model. In this model, two copies of the same data were kept in two different locations, one being the master and one the replica. Changes made to the master would then be propagated eventually over the available WAN to the replica which would eventually be up to date with the master.

What was generally impossible was to copy the whole master across the network. Instead, changes to the master would be journaled in a log as they were made. The changes would then be sent over the network to its copy of the journal. Finally, the changes in the replica journal are applied to bring the replica up to date.



Multi

Key Features of MultiSite

- *Replication:* MultiSite replicates VOBs across different geographical locations, allowing each site to have its local copy of the repository.
- *Synchronization:* Changes made to VOBs at one site are periodically synchronized with other sites. This ensures that all sites have access to the latest updates.
- *Conflict Resolution:* MultiSite includes mechanisms for detecting and resolving conflicts that might occur when multiple sites make changes to the same files or directories.
- *Site Autonomy:* Each site operates independently, meaning that even if the network connection between sites is slow or temporarily unavailable, work can continue without interruption.
- *Network Efficiency:* By keeping local copies of VOBs, MultiSite reduces the need for constant remote access, which improves performance and reduces network load.

Motivations for Using MultiSite

- *Geographical Distribution:* MultiSite is essential for organizations with development teams spread across different locations or countries. It ensures that all teams have access to the same versioned data without relying solely on a central repository.

- *Improved Performance:* Local copies of VOBs mean that developers can access and modify files quickly without experiencing the latency that might come from accessing a remote server.
- *Continuous Availability:* Even if the network connection between sites is disrupted, each site can continue to work independently. This ensures continuous development without downtime.
- *Collaboration:* MultiSite enables better collaboration among distributed teams by ensuring that everyone has access to the latest versions of files and can see the changes made by others.
- *Scalability:* As organizations grow and establish new development sites, MultiSite allows them to scale their ClearCase environment effectively without compromising on performance or data integrity.
- *Disaster Recovery:* With replicated VOBs across multiple sites, organizations can recover more easily from site-specific disasters. If one site goes down, the replicated data at other sites ensures that critical versioned data is not lost.

Note that the only thing that is being replicated is the actual VOB data and metadata. Views and many of the working files are not replicated. The rationale for his is that it is the VOBs that contain the actual corporate assets, the rest of ClearCase is just tooling to support working with those assets.

ClearCase Network

A ClearCase administrator has three principal concerns:

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 may be used only to store version-controlled data; another may 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.
- VOB or view hosted on UNIX can have auxiliary data storage on other UNIX hosts where ClearCase is not installed; such storage is accessed through UNIX symbolic links, which are not a feature of Windows NT.
- For many organizations, the set of all VOBs constitutes a central data repository, which may need to be administered as a unit.
- Most views are used by individuals; however, it is likely 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 a member of 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 not 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 a ClearCase host may play different roles at different times or several roles at once.

License server hosts.

- One or more hosts in the network act as ClearCase licence server hosts, authorizing and limiting learCase use according to the terms of your licence agreement.
- Each host on which ClearCase is installed is assigned to a particular licence server host and communicates with that host periodically.
- The `albd_server` process on a licence server host acts as the licence server process.

Registry server host.

- One host in the network acts as the ClearCase registry server host.
- Each host is assigned to a particular registry server host at ClearCase install time.
- 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 to determine the actual storage location of ClearCase data.
- The `albd_server` process on the registry server host acts as the registry server process. network with multiple registry hosts, maintaining separate clusters of
- A network can have multiple ClearCase hosts that, in general, do not share VOBs and views. Because this approach complicates the administration process, it is not recommended unless specific circumstances require it.

Backup registry server host.

- On the primary registry host, another host can be designated as a backup registry server.
- A backup registry host takes periodic snapshots of the ClearCase registry files on the primary registry server.
- If the primary server fails, registry server functions can be switched to the backup host
- Do not designate a backup registry host that is unsuitable to serve as primary registry host in an emergency.

Client hosts.

- Each user typically has one workstation called a client host, because it runs ClearCase client programs
- These programs are installed in `ccase-home-dir/bin`, including `cleartool`, `clearmake`, and various graphical user interfaces (GUIs).
- ClearCase must be installed on each client host
- Installation can include the ClearCase MVFS, which extends the native file system to provide support for ClearCase dynamic views.
- If a client uses only snapshot views, it does not need the 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 like `albd_server`, `vob_server`, `view_server`, and other programs.
- ClearCase must be installed on each server host.

Networkwide UNIX release host.

- In a network that includes UNIX ClearCase client hosts, one host in the network acts as the networkwide release host.
- This host stores an entire ClearCase UNIX release, exactly as it is supplied on the distribution medium.
- Note that this release area is active storage, not archival storage; some individual developers' workstations may access ClearCase programs and/or data through UNIX symbolic links to the release area.

Non-ClearCase UNIX hosts.

- ClearCase might not be able to be installed on every host in a network.
- It may not be possible to install it on hosts whose architectures ClearCase does not support.
- Such hosts cannot run ClearCase programs, but they can access ClearCase data through standard UNIX network file system facilities.

Versioned Object Bases (VOBs)

The network's permanent ClearCase data repository is a centralized resource. Typically, however, this 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 file-system objects and an embedded database.

Administration of a network's VOBs falls into these broad areas:

Registration.

- All VOBs are listed in a networkwide storage registry.
- In a typical network, registry maintenance is minimal; certain ClearCase commands update the registry automatically.
- The registry must be updated manually when moving a VOB to another location (for example, to another host).
- The registry may also need to be updated if different pathnames must be used on different hosts to access the same VOB storage directories.

Backup.

- VOBs have special backup and recovery requirements, and must be backed up frequently and reliably. ClearCase does not include data-backup tools; system-supplied or third-party tools are used.

Periodic maintenance.

- Administering the central repository requires balancing the need to preserve important data with the need to conserve disk space.
- ClearCase includes tools for collecting data on disk space used and for occasional scrubbing of unneeded data - what is unneeded can be specified on a per-VOB basis.
- ClearCase includes a job scheduler that manages periodic execution of various administrative tasks, including disk-space data collection and VOB scrubbing.
- ClearCase installation sets up a predefined schedule of these maintenance operations, which can be changed as needed.

Access control.

- Each VOB has an owner, a primary group, a supplemental group list, and a protection mode.
- Together, they control access to VOB data.
- As the organizational structure changes, for example, when a new project begins, there may be a need to adjust a VOB's group list.

Growth.

- As new projects begin or existing projects are placed under ClearCase control, new VOB need to be created
- This requires defining VOB access rights of various groups, and incorporating the new VOBs into the data backup and periodic maintenance schemes.

Reformatting.

- A new ClearCase release may include a feature that requires reformatting of the existing VOBs.
- This process updates the schema of the embedded VOB database.

Views

ClearCase views provide short-term storage for data created during the development process. 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: data resides in a VOB and is accessed through a view.

From an administrator’s standpoint, views and VOBs are quite similar. Each view is implemented as a view storage directory (actually a directory tree), which holds an embedded database.

- In a dynamic view, the view storage directory also holds developers’ file-system objects.
- A snapshot view has a separate snapshot view root directory that holds copies of files maintained in VOBs.

View administration is similar to that for VOBs, including registration, backup, periodic maintenance, and access control.

A view also includes both a storage area for file-system data and an associated database:

- n a dynamic view, the view’s private storage area (subdirectory .s) holds all view-private objects. It also holds the data files (data containers) for derived objects built in the view.
- In a snapshot view, view-private objects reside in the view’s directory tree.

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 tend to be shorter-lived structures, created by individual developers.

ClearCase User Base

ClearCase does not maintain a registry of its users. Any user who is logged in to a ClearCase host and can acquire a licence is able to use the software. Successful use of ClearCase depends on networkwide consistency in the user base:

UNIX users must have the same user IDs and group IDs on all hosts; Windows user and group names must be the same on all hosts. Consistency is usually achieved by using networkwide databases maintained by the operating system such as the NIS passwd and group maps on UNIX or, on Windows, Windows NT domains.

Each user has a user ID, a principal group ID, and a supplementary list of group IDs. 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 non-ClearCase commands and programs that access ClearCase data in a dynamic view must also go through the ClearCase MVFS and are 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:

1. The *object registry* records the physical storage locations of VOBs and views.
2. The *tag registry* records globally valid pathnames to VOBs and views.

For example:

- Sn object registry entry may record the fact that a VOB's storage directory is located on host `Calvin`, at pathname `C:\vobstore\project1.vbs`
- A corresponding tag registry entry may record the fact that on each developer's workstation, the VOB is to be activated (mounted) at the location specified by its VOB-tag, in this case `\proj1`.

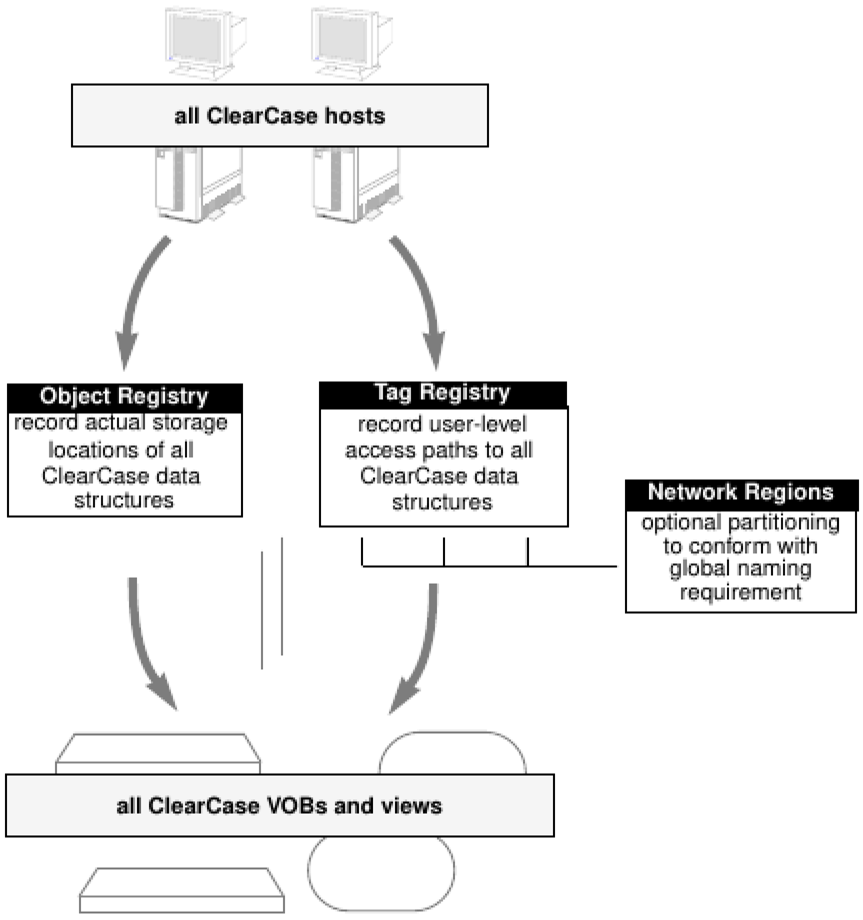
- Similarly, an object registry entry may indicate that a view storage directory is `C:\shared\integ.vws` on host Hobbs; a corresponding tag registry entry enables developers to access the view using the view-tag integration.

Network Regions

In an ideal network, all hosts access ClearCase data storage areas using identical global pathnames. Many networks, particularly those with mixed machine architectures and operating system software, fall short of this ideal.

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.


all ClearCase hosts
Object Registry
record actual storage locations of all ClearCase data structures
Tag Registry
record user-level access paths to all ClearCase data structures
Network Regions
optional partitioning to conform with global naming requirement
all ClearCase VOBs and views

Network regions

Networkwide registries offer these administrative benefits:

- Centralized control over all components of the network's distributed data repository (VOBs and views)
- Independence from operating-system-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

ClearCase Client/Server Processing

Because ClearCase is a distributed client/server application, multiple processes, running on multiple hosts, play a role in the execution of even the simplest ClearCase commands.

For example, this is what happens when a user checks out a file element:

1. The user's client process—cleartool, for example, or a GUI—issues a checkout request, in the form of a Remote Procedure Call (RPC).
2. The checkout procedure requested in the RPC is handled by several server processes, which run on the host where the VOB in which the checkout-requested element resides.
3. A view-private copy is made of the version being checked out. Making this copy involves the view_server process that manages the user's view. It can also involve other hosts:
    - The user's client process and the view may be on different hosts.
    - The VOB storage pool that holds the version being checked out may be located on a different host from the VOB storage directory.
    - On UNIX, the view may have a private storage area located on a different host from the view storage directory.

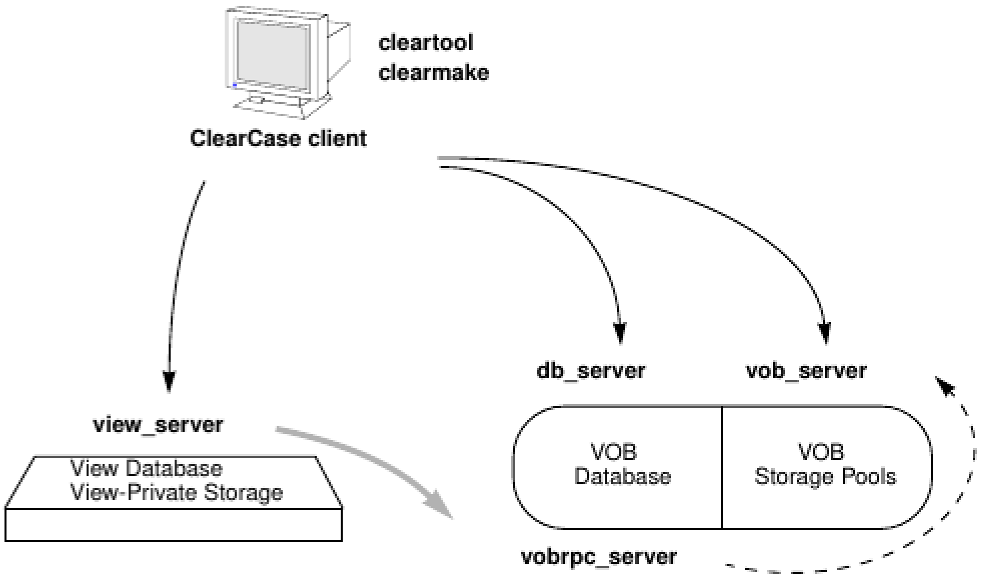
ClearCase server processes handle all this automatically and reliably. Users need not be concerned with server-level processing.

ClearCase Servers

Each ClearCase host runs an *Atria Location Broker Daemon* process, `albd_server`, which is usually started at boot time, though it can be started and stopped manually. Other ClearCase server processes are started, as needed, by the `albd_server` process.

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. ClearCase has the following servers of this type:

1. `view_server` Manages the view storage directory and database of a particular view
2. `vob_server` Manages the storage pools of a particular VOB
3. `db_server` Fields requests from one ClearCase client program, destined for one or more VOB databases on a host
4. `vobrpc_server` Fields requests from one or more view_server processes, destined for a particular VOB database



Processes

Server Error Logs

Each ClearCase server maintains an error log on the host where it executes. The actual location of these logs has changed over time but are available on both UNIX and in Windows environments.

Given ClearCase's distributed architecture, an error resulting from a command entered on one host can generate an error log entry on another host. In such cases, the user is directed to the appropriate log file on the appropriate host.

ClearCase Startup and Shutdown

ClearCase is normally started and stopped when a ClearCase host starts up or shuts down, using the normal conventions for the platform type.

MVFS

The MVFS is an extension to the native operating system.

- Most access by client programs to ClearCase VOBs or views goes through the host's MVFS when using dynamic views.
- File-system objects stored in views and VOBs are called MVFS objects.
- On UNIX computers, code that implements the MVFS is linked with a host's operating system.
- How the MVFS is linked depends on the type and version of the operating system.

If a user references a pathname.

- MVFS, which processes all pathnames within VOBs, passes the pathname on to the appropriate view_server process.
- However, before accessing ClearCase data using the MVFS, a view has to be activated and one or more VOBs mounted
- The VOB is mounted as a file system of type MVFS.

The MVFS is designed to present a selected combination of local and remote files as if they were stored in the native file system when accessed by the user. The selected files are versions of VOB files and view-private files.

The MVFS supports the following file types:

- Files
- Directories
- Symbolic links

Other file types cannot be created within a VOB.

During build script execution, `clearmake` works with the MVFS to audit low-level system calls performed on ClearCase data, recording every instance when a file or directory is created, deleted, or opened for read access. Calls involving these objects are monitored:

- Versions of elements used as build input
- View-private files used as build input—for example, the checked-out version of a file element
- Files created within VOB directories during the build

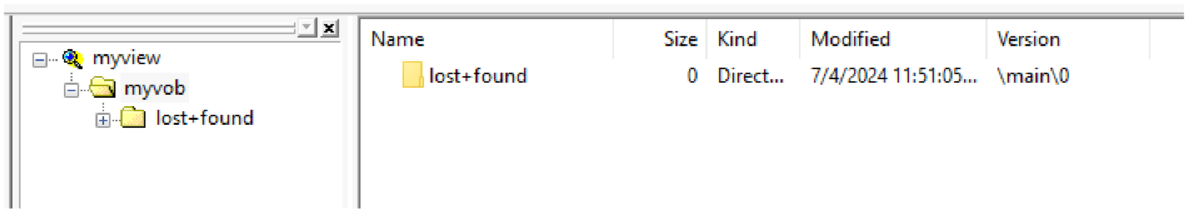
If the MVFS does not support all file-system semantics needed in a build environment, using snapshot views as an alternative.

VOB Structure

The VOB Directory

- A VOB is physically a directory in the filesystem with the extension .vbs
- It has an internal directory structure as shown below along with some administrative files.
- For example, consider a newly created VOB \myvob, from the file system view, it appears to ba an empty directory with a standard UNIX lost+found directory.

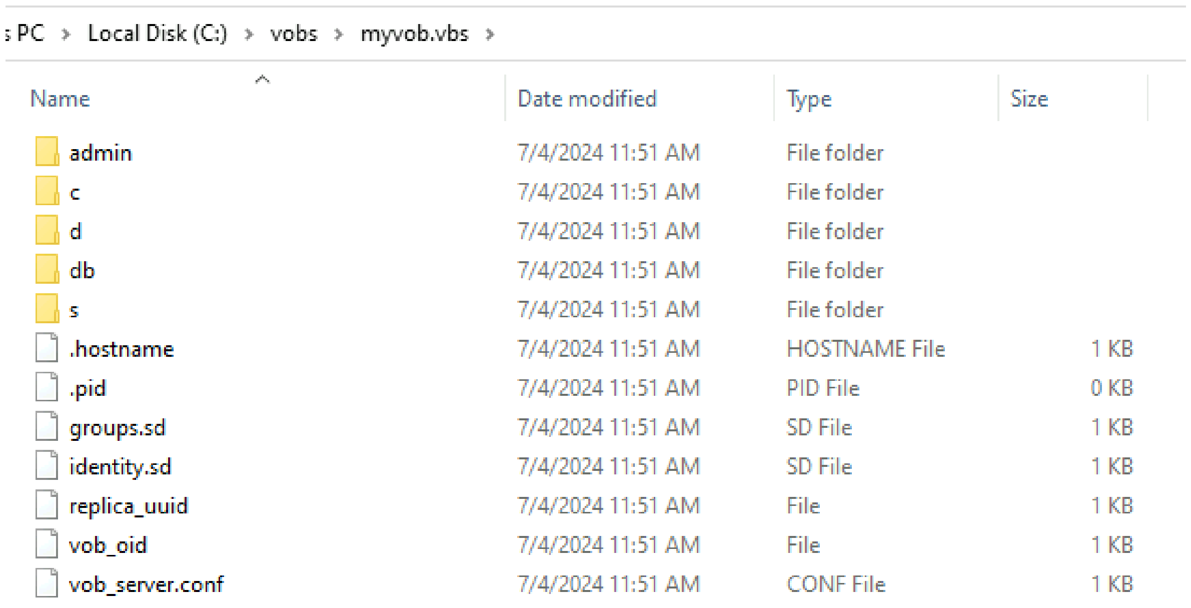
```
Y:\myvob>dir
 Volume in drive Y is CCase
 Volume Serial Number is 0234-5789

 Directory of Y:\myvob

07/04/2024  11:51 AM    <DIR>          .
07/04/2024  01:12 PM    <DIR>          ..
07/04/2024  11:51 AM    <DIR>          lost+found
               0 File(s)              0 bytes
               3 Dir(s)  52,428,800,000 bytes free
```



Name	Size	Kind	Modified	Version
lost+found	0	Direct...	7/4/2024 11:51:05...	\main\0

Empty vob

- However, the actual VOB directory looks like this:


PC > Local Disk (C:) > vobs > myvob.vbs >

Name	Date modified	Type	Size
admin	7/4/2024 11:51 AM	File folder	
c	7/4/2024 11:51 AM	File folder	
d	7/4/2024 11:51 AM	File folder	
db	7/4/2024 11:51 AM	File folder	
s	7/4/2024 11:51 AM	File folder	
.hostname	7/4/2024 11:51 AM	HOSTNAME File	1 KB
.pid	7/4/2024 11:51 AM	PID File	0 KB
groups.sd	7/4/2024 11:51 AM	SD File	1 KB
identity.sd	7/4/2024 11:51 AM	SD File	1 KB
replica_uuid	7/4/2024 11:51 AM	File	1 KB
vob_oid	7/4/2024 11:51 AM	File	1 KB
vob_server.conf	7/4/2024 11:51 AM	CONF File	1 KB

Empty vo bdir

The vob_oid file

The *vob_oid* (VOB Object Identifier) is a unique identifier assigned to each VOB.

```
C:\vobs\myvob.vbs>type vob_oid
a06a5ca3.c66c41c0.a884.5d:3b:9e:cc:47:98
```

- The vob_oid ensures that each VOB has a unique identifier, which helps in distinguishing between different VOBs in a multisite environment or within a single site.
- In a multisite ClearCase setup, the vob_oid is crucial for VOB replication. It helps in identifying and synchronizing the same VOB across different sites, ensuring that replicas are accurately matched and updates are properly applied.
- The vob_oid plays a role in maintaining the integrity of the VOB data. Since each VOB has a unique identifier, it prevents conflicts and ensures that operations are performed on the correct VOB.

The replica_uuid file

- The replica_uuid file contains a universally unique identifier (UUID) that identifies the specific replica of the VOB.
- This UUID ensures that each replica of the VOB can be uniquely identified across different sites in a MultiSite environment.
- When changes are made to the same object in different replicas, ClearCase uses the UUID to identify the source of each change and to resolve conflicts according to predefined policies.

```
C:\vobs\myvob.vbs>type replica_uuid
f6ae98e6.f7b44f62.be1d.0e:6b:8a:ef:f8:b4
```

Process files

- Recall that each VOB runs a VOB process. The `.pid` file is the process lock for the currently running VOB process.
- One of the main functions of the `.pid` file is to ensure that only a single VOB process is accessing the data in the VOB at a time. Multiple VOB processes accessing the same VOB could lead to data corruption
- The `.hostname` file references the host on which the VOB is located and helps ensure cross-host replication is done correctly in MultiSite setup.
- The `vob_server.conf` contains any custom configurations required when starting the VOB process for this VOB

Security Descriptor files

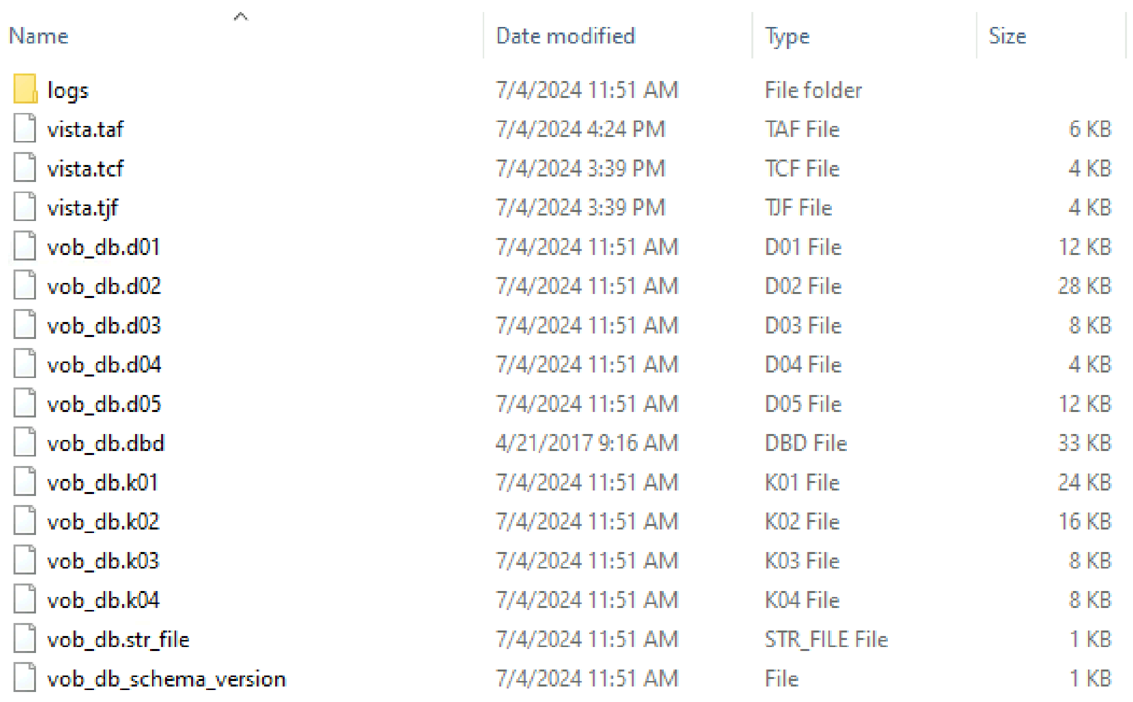
- The `groups.sd` and `identity.sd` files contain the data needed to create a security profile for the VOB that integrates with CC specific and operating system security mechanisms and ACLs.

The VOB Subdirectories

A VOB itself exists as several directories in the file system which include the following:

The VOB database

- The db subdirectory contains the binary files managed by an embedded database management system (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
    - *Metadata 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 metadata
    - Actual file-system data (for example, the contents of version 3 of file msg.c) is **not** stored in the VOB database.
- The VOB database is implemented as a set of container files in the db directory



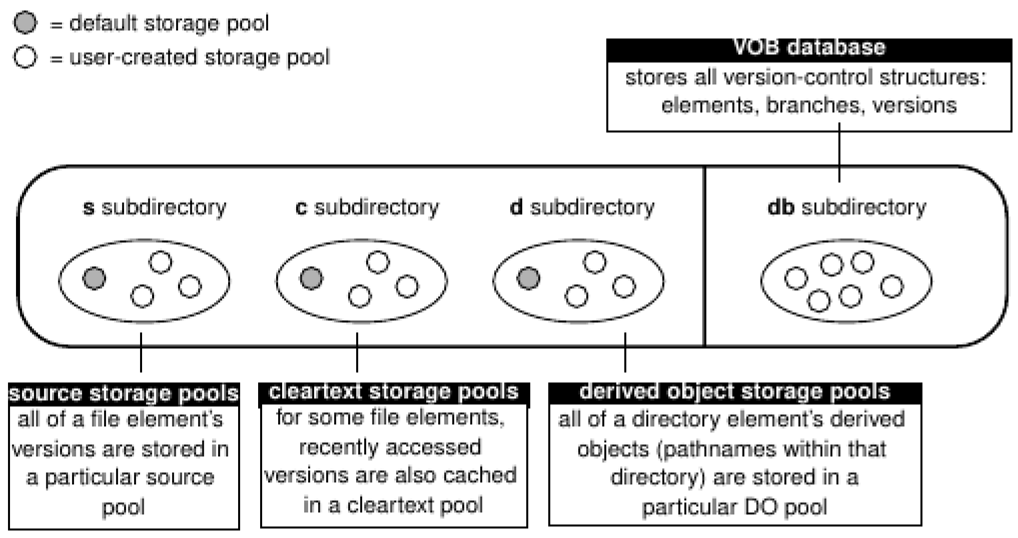
Name	Date modified	Type	Size
logs	7/4/2024 11:51 AM	File folder	
vista.taf	7/4/2024 4:24 PM	TAF File	6 KB
vista.tcf	7/4/2024 3:39 PM	TCF File	4 KB
vista.tjf	7/4/2024 3:39 PM	TJF File	4 KB
vob_db.d01	7/4/2024 11:51 AM	D01 File	12 KB
vob_db.d02	7/4/2024 11:51 AM	D02 File	28 KB
vob_db.d03	7/4/2024 11:51 AM	D03 File	8 KB
vob_db.d04	7/4/2024 11:51 AM	D04 File	4 KB
vob_db.d05	7/4/2024 11:51 AM	D05 File	12 KB
vob_db.dbd	4/21/2017 9:16 AM	DBD File	33 KB
vob_db.k01	7/4/2024 11:51 AM	K01 File	24 KB
vob_db.k02	7/4/2024 11:51 AM	K02 File	16 KB
vob_db.k03	7/4/2024 11:51 AM	K03 File	8 KB
vob_db.k04	7/4/2024 11:51 AM	K04 File	8 KB
vob_db.str_file	7/4/2024 11:51 AM	STR_FILE File	1 KB
vob_db_schema_version	7/4/2024 11:51 AM	File	1 KB

Vobdb files

- The `vob_db.dxx` files contain the actual database contents.
- The `vob_db.kxx` files contain indexing information
- The `vists.*` files are used to improve the performance of database queries. They contain precomputed information that speeds up the retrieval of data from the VOB's database.
- The
- ClearCase server programs are invoked as needed to access a VOB’s database:
- These server processes run on the host where the VOB storage directory physically resides.
- The db subdirectory must also be on that same host.
-

VOB Storage Pools

- The c, d, and s subdirectories contain the VOB’s storage pools, each of which is a standard 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 may be stored in separate data container files, or may be combined into a single structured file that contains deltas (version-to-version differences).


● = default storage pool
○ = user-created storage pool
VOB database
stores all version-control structures: elements, branches, versions
s subdirectory
c subdirectory
d subdirectory
db subdirectory
source storage pools
all of a file element's versions are stored in a particular source pool
cleartext storage pools
for some file elements, recently accessed versions are also cached in a cleartext pool
derived object storage pools
all of a directory element's derived objects (pathnames within that directory) are stored in a particular DO pool

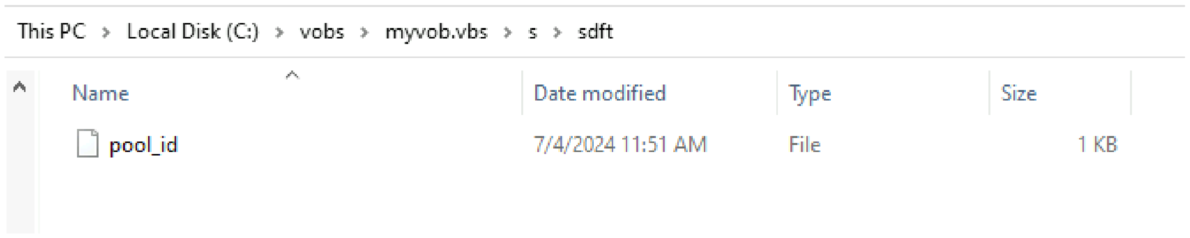
Storage pools

The contents of the pools is a set of container files organized for fast access and updating.

Source Storage Pools

- Each source pool holds all the source data containers for a set of file elements.
- A source data container holds the contents of one or more versions of a file element.
- 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.
- The default source storage pool is located in the directory `sdft`
- Source pools are accessed by cleartool commands such as `checkout` and `checkin`, as well as by system utility programs like `cat` or that read the contents of elements that are not checked out.

- If a cleartext version of the element is available, it is used. If it is not available, the request to access the file element causes the cleartext to be constructed and stored in the cleartext pool.
- For example, this is the default source storage pool for an empty vob

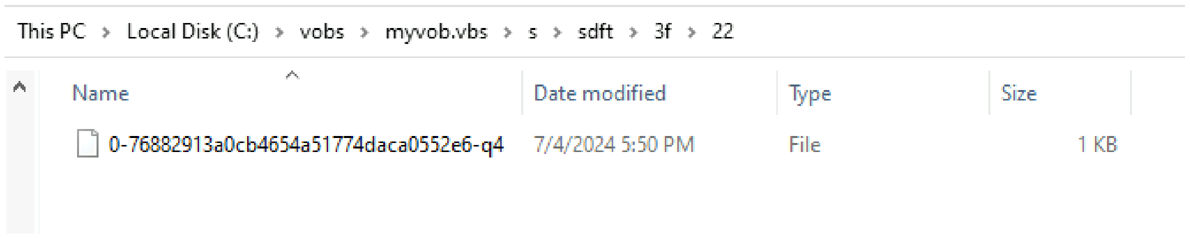


This PC > Local Disk (C:) > vobs > myvob.vbs > s > sdft

Name	Date modified	Type	Size
pool_id	7/4/2024 11:51 AM	File	1 KB

Emptyspool

- After a file has been placed under version control, it looks like this



This PC > Local Disk (C:) > vobs > myvob.vbs > s > sdft > 3f > 22

Name	Date modified	Type	Size
0-76882913a0cb4654a51774daca0552e6-q4	7/4/2024 5:50 PM	File	1 KB

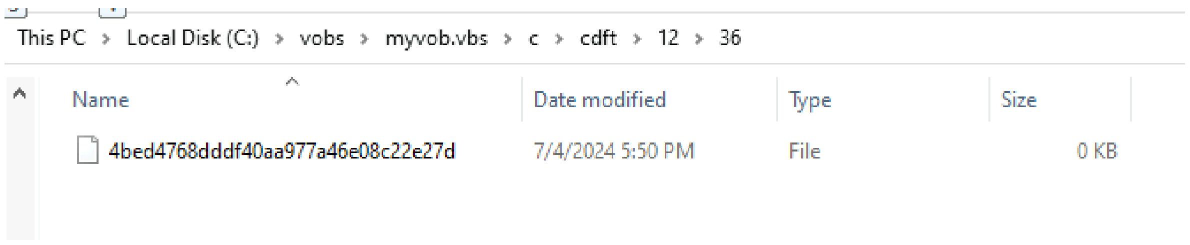
Onefilespool

Cleartext Storage Pools

- Each cleartext 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 speed up access to elements for which all versions are stored in a single data container which are compressed text 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 later 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, because cleartext storage pools are periodically scrubbed.
- The default storage pool is `cdft`

- The image below shows the Cleartext pool after an element has been accessed


This PC > Local Disk (C:) > vobs > myvob.vbs > c > cdft > 12 > 36

Name	Date modified	Type	Size
4bed4768dddf40aa977a46e08c22e27d	7/4/2024 5:50 PM	File	0 KB

Cleartext

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 (usually binary data) of one DO, created by a ClearCase build tool (`clearmake`, for example, or `clearaudit`).
- The data containers for unshared and non shareable derived objects reside in view-private storage
- Winkin operations copy the DO to the VOB so that it can be reused in other views.
- DO storage pools contain data containers only for the derived objects that have been copied to the VOB through ClearCase's winkin feature.
- Each directory element is assigned to a particular DO storage pool.
- The first time a DO created within that directory is winked in, its data container is copied to the corresponding DO storage pool.
- By default, derived object pools are periodically scrubbed to remove extraneous DOs

Setting Up VOBs

A ClearCase data repository is a set of globally accessible VOBs. The major steps to setting uo a VOB are

1. Select a host for a new VOB.
2. Modify operating system resources, if necessary.
3. Create one or more VOB storage locations if necessary
4. Create the VOB
5. If necessary, adjust its registry and identity information to ensure global accessibility and implement access controls.
6. Coordinate the VOB with other existing VOBs.
7. Populate the VOB with new or existing development data.

After VOBs are set up, they can be activated for use with dynamic views with the cleartool `mount` command, or from the VOB Admin Browser on UNIX, or on Windows with the ClearCase Explorer or Windows Explorer.

Two common user mistakes users make that can make a VOB invisible to their dynamic views are:

1. forgetting to mount the VOB and
2. forgetting to work in a view.

Users of snapshot views must load the view from a VOB before they can access VOB data.

By default, UNIX hosts mount all public VOBs at ClearCase startup time.

To arrange for a Windows host to mount a VOB each time at log on:

- Select the *Reconnect at Logon* check box in the Mount dialogue box when the VOB is first mounted.

- Use the `–persistent` option to cleartool `mount`.
- Add a cleartool `mount` command to a .BAT file in the Startup program group.

Selecting a VOB Host

A host on which one or more VOB storage directories reside is a VOB host. A typical network distributes its VOBs among several VOB hosts. Nearly any host supported by ClearCase can be a VOB host, although UNIX is usually preferred.

The computer chosen as a VOB host must satisfy these requirements:

Main memory (RAM).

- The minimum recommended main memory size is at least 128 MB or half the size of all the VOB databases the server will host, whichever is greater.
- To this amount should be added 7 MB of memory per VOB regardless of VOB database size, as well as 750 KB of memory for any view_server process that will run on the VOB host.
- Adequate physical memory is the most important factor in VOB performance; increasing the size of a VOB host's main memory is the easiest and the most cost-efficient way to make VOB access faster and/or to increase the number of concurrent users without performance degrading.
- For the best performance, configure the VOB host with enough main memory to hold the entire VOB database.

Disk capacity.

- A VOB database must fit in a single disk partition
- VOB databases tend to grow significantly as development proceeds and projects mature.
- If possible, use a RAID or similar disk subsystem that takes advantage of disk striping and mirroring.

- Mirrors are useful for backups, although there is a slight performance degradation associated with their use. However, striping helps overall performance and more than makes up for any degradation caused by mirroring.

Creating VOB Storage Locations

ClearCase allows the specification one or more server storage locations for each network region to be defined in the ClearCase registry.

- A server storage location has to be specified for VOB or view storage when created.
- VOB storage locations provide administrators with a way to designate specific hosts and disks that will be recommended to users or other administrators creating new VOBs.
- Although these storage locations are not the default choice for VOB-creation tools, creating and using them can simplify many VOB setup and administration tasks.
- VOB storage locations should be created on a disk partition that has plenty of room for VOB database growth and is accessible to all ClearCase client hosts in the region.
- On UNIX hosts, the partition must be exported so that other UNIX clients can mount it.
- For Windows hosts, the directory (folder) must be shared.

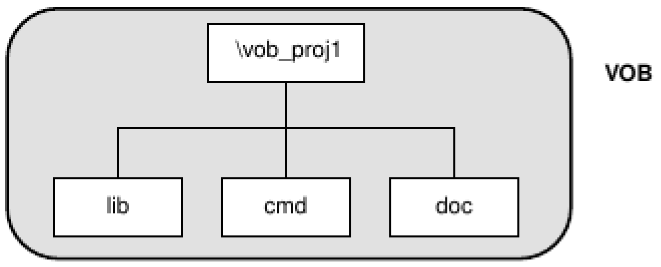
Planning for One or More VOBs

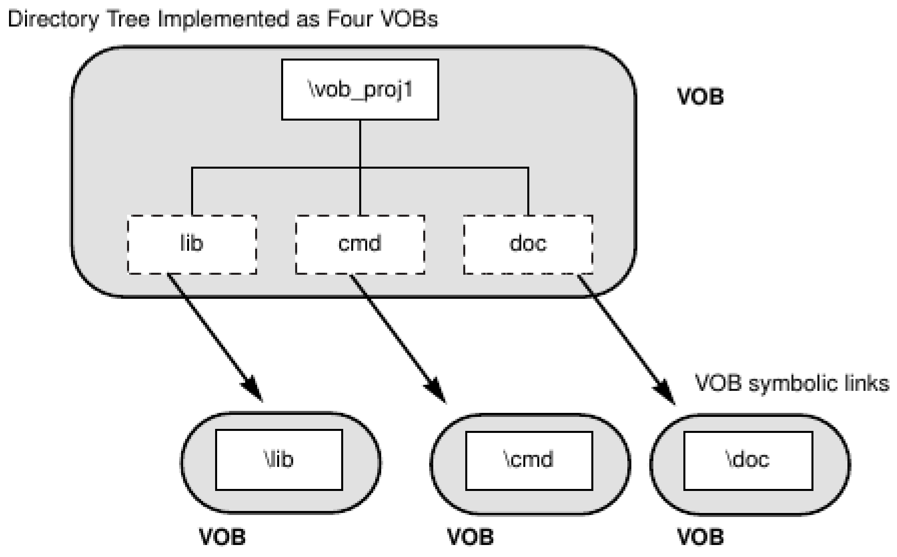
- It is necessary to decide how to allocate data to one or more VOBs on a VOB host.
- These are the principal trade-offs to consider:
  - Splitting data into several smaller VOBs increases your flexibility.
  - It is easy to move a single VOB to another host but more difficult to split a VOB into several new VOBs and move one or more of them to another host.
- Typically, there are fewer performance bottlenecks when using several smaller VOBs rather than one very large VOB.
- Having fewer VOBs facilitates data backup.

- Having fewer VOBs facilitates synchronizing label, branch, and other definitions across all the VOBs. It reduces reliance on the availability of admin VOBs and globally defined metadata types.

To users, a VOB appears to be a single directory tree.

- It makes sense to consider organizing the development artifacts into logically separate trees and create a VOB for each one.
- If two projects do not share source files, the source files can be in different VOBs.
- Typically, several or all projects share some header (.h) files which could be shared sources in their own VOB.
- In VOB planning, keep in mind that you can make several VOBs appear to be a single directory tree, using VOB symbolic links.





Vobtree

- Be sure that the text of a VOB symbolic link is a relative pathname, not a full or absolute pathname.
- For example, the following command creates a VOB symbolic link that makes the VOB `\vob_lib` appear as a subdirectory named `lib` in the VOB `\vob_proj1`:

```
cleartool ln -s ..\vob_lib lib     _(..\vob_lib, not \vob_lib)_
Link created: "lib".
cleartool ls lib
lib --> ../lib
```

Relative pathnames ensure that the link is traversed correctly in all view contexts

Release VOBs

- VOBs are not for source files only; but can be used to store product releases (binaries, configuration files, bitmaps, and so on).
- Such VOBs tend to grow quickly.
- Releases for different platforms should be stored in separate VOBs.

Creating a VOB

To create a new VOB:

Log on to the VOB host

- The host should meet the criteria laid out in *Selecting a VOB Host* earlier
- If possible, log on as a user who is a member of the same primary group as all other users who will need access to the VOB.
- The identity of the user who creates a VOB (user ID, primary group, and umask on UNIX; user ID and primary group on Windows NT) is used to initialize VOB access permissions.
- If a VOB is created by a user whose primary group is different from the primary group of other users who must access the VOB, the VOB's supplementary group list will need to be edited before those users can access VOB data.

Choose a location for the VOB storage directory.

- An existing server storage location can be used or a new one created
- Any other directory that has the proper characteristics for good VOB storage can be used, but this is not recommended

Choose a VOB-tag.

- A VOB-tag is a globally unique name by which all clients can refer to a VOB.
- A VOB-tag should indicate what the VOB contains or is used for.
- VOB tags exhibit syntactic differences on UNIX and Windows hosts because they must conform to the network naming conventions of each platform.
- On UNIX hosts, each ClearCase client mounts the VOB as a file system of type MVFS, so the VOB-tag must be the full pathname of a mountable file system, for example, `/vobs/flex`
- This pathname usually includes a local mount-point (`/vobs`) and the name of a mountable file system (`/flex`).
- Unless there is a compelling reason to do otherwise, all clients should mount the VOB at the same local mount point.
- On Windows hosts, each ClearCase client host uses the VOB-tag as though it were the name of a Windows directory.
- In a Windows network, a VOB-tag is the VOB's registered name and also its root directory.
- A Windows VOB-tag must have exactly one backslash (\), which must be the first character of the VOB-tag (for example, `\vob_project2`)
- When activated with the cleartool mount command, a VOB-tag appears as a subdirectory under each view-tag visible on the dynamic-views drive (M: by default).

Create the VOB.

- This example explains how to create a VOB using the cleartool command line. You can also create VOBs using various GUI tools on UNIX or Windows.
- The following command creates a VOB on Windows with the VOB-tag `flex` whose storage is in the directory shared as `vobstore` on a host named `pluto`, then returns location and ownership information about the newly created VOB.

```
Host-local path: c:\vobstore\flex.vbs
Global path: \\pluto\vobstore\flex.vbs

VOB ownership:
owner vobadm
group dvt
```

- Whenever a VOB is created, the user is prompted to enter a comment, which is stored in an event record `create versioned object base”` in the new VOB’s database.
- By default, VOBs are created as private VOBs.
- Specifying `-public` on the `mkvob` command line makes a public VOB.
- Public VOBs can be mounted one at a time using the cleartool mount command, or as a group using the command cleartool `mount –all`.
- If `-public` is specified on the `mkvob` command line, the user is prompted to enter the ClearCase registry password.
- This password is normally established when ClearCase is installed at a site.

Coordinating the New VOB with Existing VOBs

- A typical project involves multiple VOBs.
- To ensure that they all work together, their type objects: branch types, label types, and so on must be coordinated.
- For example, the following config spec can be used to create a view that selects the source versions for a previous release:
-

```
element * REL_3
```

- For this strategy to succeed, all relevant VOBs must define version label `REL3`; that is, label type `REL3` must exist in each VOB
- If using global types, create the label type with the `–global` option to `mklbtype`.
- The VOB that stores this global type becomes, by definition, an administrative VOB.
- Users in other VOBs then link to the administrative VOB by creating instances of the predefined hyperlink type AdminVOB that point to the administrative VOB.
- Thereafter, users in any client VOB can create instances of label REL3.

Upgrading a VOB to a New Release

New ClearCase releases often introduce new features that require changes to the VOB database schema or other internal VOB data structures. To use these new features, admins may need to take one or both of two actions to upgrade an existing VOB:

1. Change the VOB's feature level.
2. Reformat the VOB to change the VOB's database schema.

These actions are independent of each other.

- In general, the feature level of a VOB does not need to be raised unless it is desired that the VOB have the capabilities that require a higher feature level.
- Reformatting a VOB may be required, optional, or not applicable for a particular release.

Upgrading the Feature Level of a VOB

- A feature level is an integer that defines which ClearCase features a VOB supports.
- Whenever a ClearCase release introduces features that require support in the VOB server, an admin must raise the feature level of a VOB before clients can take advantage of the new features when accessing data in that VOB.

- The primary purpose of feature levels is to manage VOBs that are replicated (using ClearCase MultiSite) across server machines that are not all running the same ClearCase release.
- Every ClearCase release is associated with a feature level.
- The cleartool describe command shows the feature level for a VOB.
- The `chflevel` command changes the feature level of a VOB.
- To raise the feature levels of unreplicated VOBs on a VOB server host:
  1. Log on to the host that contains VOB storage directories for the VOBs to be upgraded.
  2. Execute the `chflevel` command with the `–auto` option.
  3. The command lists each VOB whose storage directory is located on the host.
  4. It then offers to raise the feature level of each unreplicated VOB that is not already at the feature level corresponding to the release of ClearCase that is installed on the host.
- When using MultiSite, each VOB replica has a feature level, and the VOB family has a feature level.
- Replicas in the same family can have different feature levels.
- The family feature level is the feature level that is equal to or less than the lowest replica feature level found among members of the VOB family.
- Before raising the feature level of a VOB family, admins must raise the feature levels of all replicas in that family.

Reformatting a VOB

- A release of ClearCase may introduce a new schema, or format, for the VOB database.
- The new format may support new product features, enhance storage efficiency, or improve performance.
- Upgrading existing VOBs to use the new database format may be required or optional

for a particular release.

- In general, all VOBs that reside on a VOB server host must use the same database format.
- To change the database formats for VOBs:

1. Back up all VOBs on the VOB server host.
2. Ensure that the ClearCase installation on the VOB server host supports the new VOB database format.
3. Run the `reformatvob` command for each VOB storage directory on the host.

- By default, the command dumps the existing VOB database into an ASCII representation, then loads that representation into a new database that uses the format corresponding to the release of ClearCase that is installed on the host.

Administrative VOBs

An administrative VOB stores definitions of global types and makes them available to all client VOBs that link to the administrative VOB. Global types can be used to increase the scope of a type object from a single VOB to a group of VOBs. For example, all the VOBs in the local area network can use the same set of type objects by linking them all to the same administrative VOB.

A client VOB uses global types from an administrative VOB as follows:

1. A developer attempts to create an instance of a type, but there is no type object in the client VOB. For example, a developer wants to create a `v3_bugfix` branch in a particular element, but branch type `v3_bugfix` does not exist in the client VOB.
2. ClearCase determines which administrative VOB is associated with the client VOB by scanning for an `AdminVOB` hyperlink in the client VOB.
3. If it finds a global type (branch type `v3_bugfix` in this example) in the administrative VOB, ClearCase creates a copy of the type object in the client VOB. This capability is called auto-make-type.

4. ClearCase uses the local copy of the type object to create the instance that the developer wants.

A local copy is linked to the global type object by a GlobalDefinition hyperlink. Operations performed on a global type affect all its local copies. Similarly, operations performed on a local copy affect the global type, and by extension, all other local copies.

Why Use Global Types?

Using global types offers the following benefits:

1. *Automatic creation of types.* Local metadata types are created from global types as needed in client VOBs when users create instances of the types with the `mkattr`, `mkbranch`, `mkelem`, `mkhlink`, and `mklabel` commands.
2. *Centralized administration.* Groups of VOBs can share metadata type definitions. Admins can control global type objects and their local copies from the administrative VOB.

Backing up and Restoring VOBs

Backing Up a VOB

Backing Up a VOB on UNIX

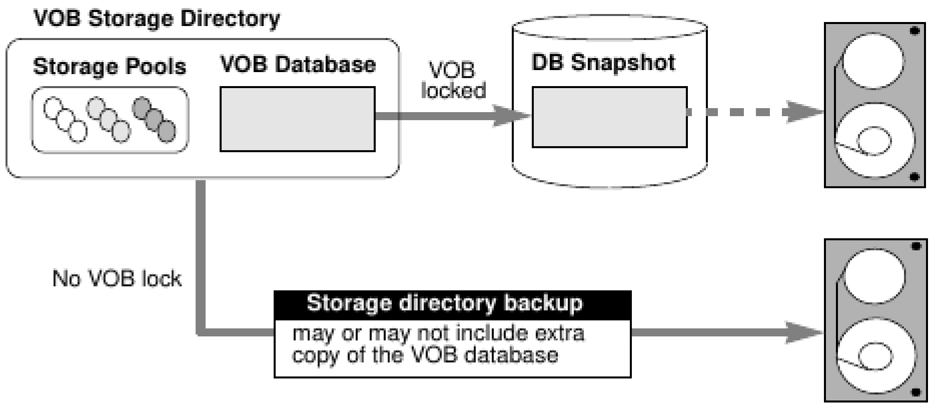
A VOB backup on UNIX can be summarized as follows:

1. Lock the VOB.
2. Back up the VOB storage directory.
3. Unlock the VOB.

Choosing Between Standard and Semi-Live Backup

The standard backup procedure is to lock the VOB and back up the entire VOB—VOB database, plus VOB storage pools and various other files in the VOB storage directory.

- The VOB must remain locked for the duration of the backup.
- Keeping VOB database and storage pools synchronized on backup media is desirable; the standard backup procedure is recommended for all sites that can accept the duration of required locks.
- Note that ClearCase build programs are designed to cope with locked VOBs. They will "sleep" when they are unable to open an object in a locked VOB, and then retry



Siemi live

- As illustrated in above, Semi-live backup involves backing up the two major pieces of a VOB separately, as follows:

1. *VOB database.* Configure the VOB to have the vob_snapshot utility periodically lock the VOB and copy the VOB database to another location on disk (from which it can be backed up as part of the site's normal backup process).
2. *VOB storage pools.* Back up the VOB storage directory routinely, without locking the VOB. The backed-up VOB storage directory may include a copy of the VOB database. However, because it is backed up while the VOB is unlocked, this database is useless and is discarded when the VOB is restored.
3. If THE UNIX VOB servers have remote storage pools, they must be backed up as well.

Benefits of Semi-Live Backup

- VOB lock time reduced.
- The VOB storage directory can be backed up while unlocked.
- The `vob_snapshot` utility locks the VOB, copies the VOB database to another disk location, and unlocks the VOB.

Costs of Semi-Live Backup

- More disk space is required. A second copy of the VOB database must be captured to disk.
- Each of the VOB's source pool data containers, when replaced by a container with new version data, is retained for an additional 30 minutes to guarantee that source containers can be reconstructed when `checkvob` resynchronizes the VOB database and the storage pools at VOB restore time.
- The VOB restore procedure is more complex.
- The VOB storage pools and the VOB database have different reference times.
- VOB database and storage pools must be resynchronized when restoring the VOB. In particular, DO and version data added or removed in the interval between the database snapshot and storage pool backup cause database or pool skew that must be resolved.
- Some data may be lost at VOB restore time.
- If the restored pools are older than the restored VOB database, data missing from the pools is lost.
- If the restored pool backup is newer than the database, pool version data newer than the snapshot is not added to the restored database.

Enabling Semi-Live Backup

- To enable database snapshots, run vob_snapshot_setup on a VOB.
- This command causes the ClearCase scheduler to run `vob_snapshot` periodically on the VOB (daily, by default).
- The backup procedures that follow accommodate (but do not require) snapshot-enabled VOBs.
- However, their focus is the standard backup approach, which requires the VOB to be locked for long enough to back up the database and all the pools.
- The vob_restore procedure applies equally to VOBs with and without database snapshots.

Ensuring a Consistent Backup

A backup represents a self-consistent snapshot of a VOB storage directory's contents only if the VOB is not modified while the backup program is working. That's why it is essential to lock the VOB before backing it up and unlock it after the backup completes.

Regardless of the chosen backup strategy, the VOB must be locked for all users.

- The lock is mandatory. It is not sufficient to capture a VOB that happens to be idle.
- The lock does more than ensure that no one modifies the VOB. It also causes the VOB to flush a database checkpoint to disk before backup.
- A VOB lock applied with the `–nusers` option does not perform the required database checkpoint.

Locking and Unlocking a VOB

- Only a privileged user can lock or unlock a VOB.
- On the command line, use cleartool lock and unlock:

```
cleartool lock vob:/vobs/flex
   Locked versioned object base "/net/pluto/vobstore/flex.vbs"
```

- Then after the backup
-

```
cleartool unlock vob:/vobs/flex
Unlocked versioned object base "/net/pluto/vobstore/flex.vbs
```

Restoring a VOB from Backup with vob_restore

- Given a complete and consistent VOB storage backup, `vob_restore` can accommodate a variety of restore scenarios.

- `vob_restore` handles all the following subtasks:

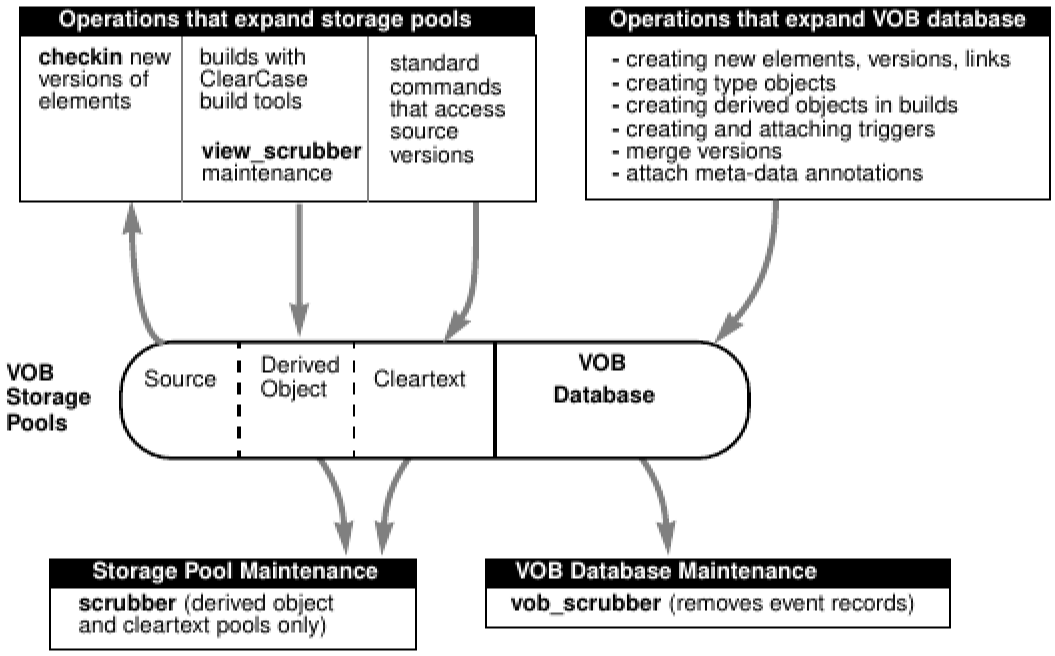
1. Stops and restarts ClearCase
2. Updates the ClearCase VOB registry
3. Merges the VOB database snapshot and VOB storage directory
4. Copies the temporary storage directory to the target location
5. Runs `checkvob` to resynchronize the VOB database and storage pools

- `vob_restore` can be run with or without a VOB snapshot. Any valid VOB backup will work. Before examining alternative restore scenarios, it is useful to summarize the restoration procedure. We recommend that you do not retrieve VOB storage from backup media until vob_restore prompts you to do so in Step #3. 1.Log on to the VOB host. Log on as a user with permission to stop and start ClearCase— typically the root user on UNIX or the local Administrator on Windows NT.

Administering VOBs

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

1. Preserving critical data and metadata.
2. Discarding data and metadata that is no longer important to minimize the disk space required.


Operations that expand storage pools
checkin new versions of elements
builds with ClearCase build tools
view_scrubber maintenance
standard commands that access source versions
Operations that expand VOB database
- creating new elements, versions, links
- creating type objects
- creating derived objects in builds
- creating and attaching triggers
- merge versions
- attach meta-data annotations
VOB Storage Pools
Source
Derived Object
Cleartext
VOB Database
Storage Pool Maintenance
scrubber (derived object and cleartext pools only)
VOB Database Maintenance
vob_scrubber (removes event records)

Vob growth

- The diagram shows how VOB storage pools and VOB databases grow in regular use;
- And lists the maintenance commands (scrubbers) that control the growth of these storage areas.
-

Scrubbing VOB Storage Pools

On any VOB host, the ClearCase scheduler runs a daily job that scrubs the storage pools of all VOBs whose storage directories reside on that host:

1. Source pools are never scrubbed automatically. Source versions are too valuable to be routinely deleted.
2. 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). Scrubbing also removes the corresponding derived objects from the VOB database.
3. Cleartext pools are scrubbed to control their size. These pools are essentially caches; scrubbing unneeded data containers in a cleartext pool has little or no effect on ClearCase performance.

Storage pools are scrubbed by the scrubber utility.

- Each derived object and cleartext storage pool has its own scrubbing parameters, which control how the scrubber processes that pool.
- To change the way VOB storage pools are scrubbed, the scrubbing parameters can be changed for an individual pool using the `mkpool –update` command.

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 may not be useful or may lose their value as time passes. (For example, the removal of an unneeded or obsolete version label.)
- Each host has a scrubber configuration file, `ccase-home-dir\config\vob\vob_scrubber_params`, which controls `vob_scrubber` operation for all VOBs on that host.
- If more control is needed over scrubbing schedules, a scrubber parameters file can be created for each VOB.
- This file is also named `vob_scrubber_params`, but it is located in the VOB storage directory.

- Deletion of event records and derived objects from a VOB database by the vob_scrubber and scrubber utilities is logical, not 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 it is needed to shrink a VOB's on-disk storage, the `reformatvob` command discards all such free space.
- It is not recommended tp shrink VOBs routinely; it's usually enough to use maintenance procedures that keep them from growing too fast.

Removing Unneeded Versions from a VOB'

In general, removal of source data from VOB storage should be done with extreme caution.

Removing entire elements, using `rmelem`, is particularly dangerous:

- Even if an element is no longer needed for the next release, it may be needed to reproduce and maintain previous releases.
- `rmelem` removes the element's name from all directory versions in which it was ever cataloged. This erasing of history means that the element does not appear in listings or comparisons of old directory versions.
- Renaming an element, using the `rmname` command, preserves its history without cluttering future projects.
- If it is necessary to reclaim disk space, it is more prudent to remove individual versions of elements, rather than entire elements.
- The `rmver` command makes it easy to remove versions that will probably never be needed again.
- By default, `rmver` removes only versions of little use:
  - 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 metadata annotations: version labels, attributes, or hyperlinks

Creating Additional Storage Pools for UNIX VOBs

If a VOB on a UNIX server is threatening to fill up its disk partition, it is possible to rework the VOB's storage pools to take advantage of remote storage.

- This feature requires UNIX symbolic links, so it's not available on Windows

The VOB database (db subdirectory) must be physically located within the VOB storage directory, but ny or all of its storage pools can be remote.

- If necessary, all the VOB's default storage pools can be abandoned to allow a transfer of all the VOB's file-system data to other disk partitions and/or remote hosts.
- The host for 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`).

When deciding which hosts to use for new storage pools, consider that each kind of storage pool has a different pattern of use:

Source pools.

- These pools store the most important data: the checked-in versions of file elements.
- Traffic involving these pools is relatively light, but data integrity is crucial.
- The ideal location for such pools is a robust file server, with a large capacity and frequent, reliable data backups.

Cleartext pools.

- These pools probably get the heaviest traffic (assuming that many of the file elements are stored in delta and/or compressed format).
- But the data in cleartext pools is expendable, because ClearCase can reconstruct it.

- The ideal location for such pools is a machine with a fast file system.

Derived object pools.

- These pools can become quite large, depending on the number of active configurations—multiple new development projects and 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.

These are the principal commands for working with storage pools:

- `lspool` Lists a VOB's existing storage pools.
- `describe` Lists an element's storage pool assignments. The extended naming symbol `@@` when specifying the element:

```
cleartool describe getcwd.c@@
```

- `mkpool` Creates a new storage pool. Use the –ln option to create a remote storage pool.
- `chpool` Reassigns 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 controls which pools are used by new file elements and derived objects created within the directory.

It is recommended that source storage pools be kept local, within the VOB storage directory. This strategy optimizes data integrity: a single disk partition contains all the VOB's essential data. It also simplifies backup/restore procedures. This concern typically overrides performance considerations because losing a source pool means that developers must re-create the lost versions.

Adjusting Storage Pool Scrubbing

Typical motivations for adjusting a VOB host's default procedures for storage pool scrubbing include:

1. *Not enough space.* The disk partitions in which VOB storage pools reside may fill up frequently. A more aggressive scrubbing strategy may be necessary.
2. *Not enough time.* The scrubber utility may be taking too much time to complete, interfering with other overnight activities, such as nightly software builds. A less aggressive scrubbing strategy may be necessary.

It may be necessary to experiment. For example, adjusting scrubbing to take place less frequently may cause disk-space problems that had not previously experienced. Before making any scrubbing adjustments on a VOB host, be sure to analyze its scrubber_log file.

If a Windows backup tool is used that changes the time stamps in VOB storage directories, the DO and cleartext pools in those directories may never be scrubbed. The scrubber command, by default, scrubs objects that have not been referenced for the last 96 hours. If such a backup tool runs every night, the access time on objects in the pools is reset every night and the objects are never scrubbed.

Moving VOBs

VOBs cannot be copied from one location to another using an ordinary file copy utility. Special procedures must be followed to maintain the integrity of VOB data and to preserve permissions and ownership on VOB storage locations.

Restrictions on Moving a VOB

In general, when you move a VOB using the procedure presented here, the user and group information in the new location must match the user and group information in the old location.

MultiSite facilities can be used to make a new replica of the existing VOB and *then delete the old replica.*

Do not use ClearCase MultiSite to create multiple replicas of a VOB in a single ClearCase region. Because the VOB UUID is identical for all replicas in a VOB family and is stored in many structures within a VOB, there is no way to make a replica unique. Creating and using multiple replicas of a VOB in a single region causes ClearCase to exhibit unpredictable behaviour, may `cause data loss, and is not supported by Rational Software.

The UNIX Move VOB Procedure

1. Determine whether the VOB has any non-local storage pools.
2. Verify the validity of pathnames to non-local pools.
3. Deactivate the VOB using the cleartool `umount` command
4. (If applicable) Disable VOB snapshots on the current host. I
5. Back up the VOB storage directory
6. Lock the VOB.
7. Move the VOB storage directory.
8. Ensure that the old VOB cannot be reactivated
9. Terminate the old VOB's server processes.
10. Register the VOB at its new location
11. Reactivate the VOB
12. Unlock the VOB.
13. Free any old or unneeded VOB storage.
14. Enable VOB snapshots on the new host.

Removing VOBs

Suppose that the VOB to be taken out of service has storage directory `/net/sol/vobstore/libpub.vbs` and has VOB-tag `/proj/libpub`.

1. Have all clients unmount the VOB.
2. Remove the VOB from the object registry.
3. (Optional) Remove the VOB from the tag registry

4. Terminate the VOB’s server processes.

View Structure

ClearCase Views

- A ClearCase development environment can include any number of views.
- A typical view is private to a single user, or small team.
- A view provides a workspace where users access versions of ClearCase elements and use other file-system objects that are outside ClearCase control
- ClearCase has two kinds of views: dynamic views and snapshot views.

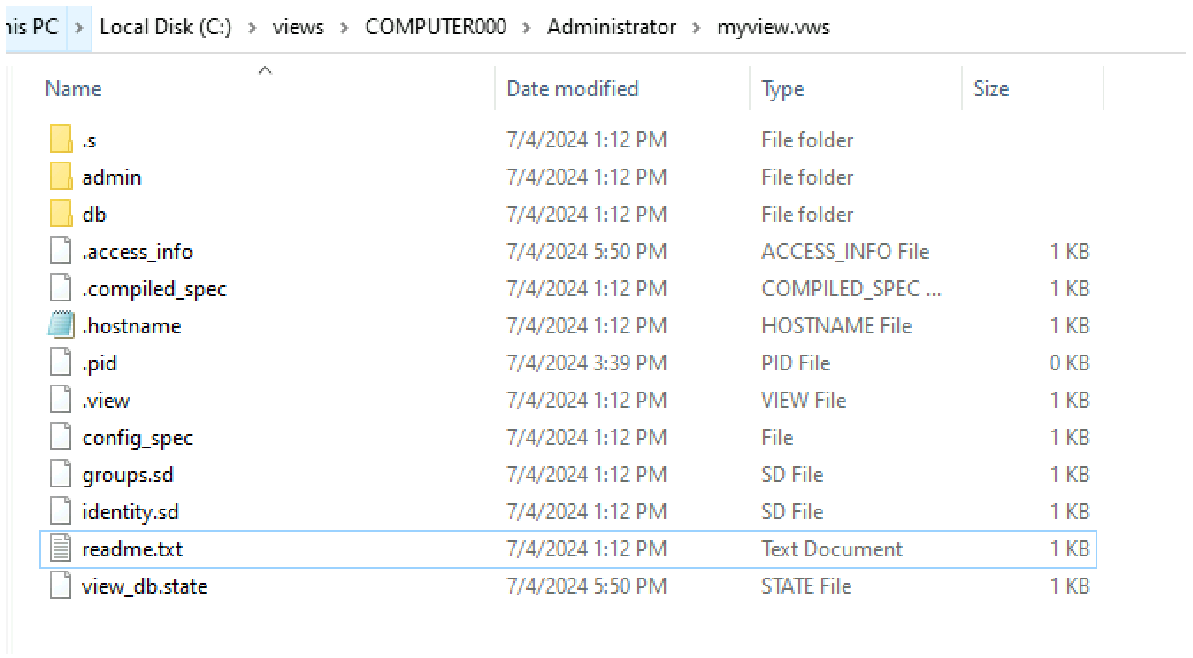
Dynamic Views

- A dynamic view provides transparent access to versions of elements.
- Each time an element is accessed, the view_server evaluates the view's config spec and selects a version of the element.
- Each view implements a virtual work area, which presents users with an extended file system that appears to be an ordinary file system hierarchy.
- The work area is implemented by MVFS.
- This work area includes these items:
  - Selected versions of elements, actually stored in VOB storage pools but displayed in the View
  - Checked out files that are being modified, 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 and stored in the view's private storage area
  - Configuration records that correspond to these derived objects

- Derived objects originally built in another view and then winked in to this dynamic view via the VOB derived oject storage pools.
- View-private objects: miscellaneous files, directories, and links that appear only in this view and are stored in the view's private storage area

View Storage

Like a VOB, a View is stored in a directory but with the `.vws` extension. Also like a VOB, this directory has a specific structure.



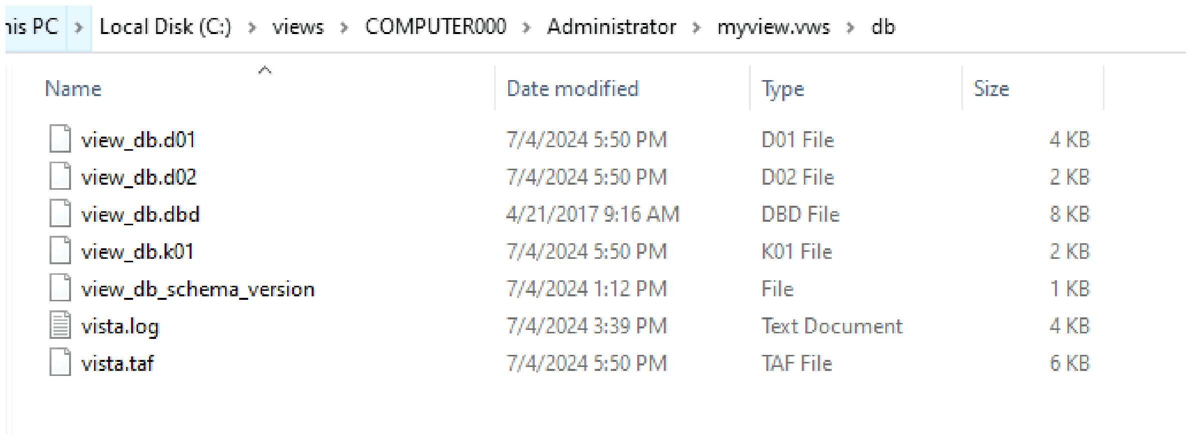
```
ıis PC > Local Disk (C:) > views > COMPUTER000 > Administrator > myview.vws
```

Name	Date modified	Type	Size
.s	7/4/2024 1:12 PM	File folder	
admin	7/4/2024 1:12 PM	File folder	
db	7/4/2024 1:12 PM	File folder	
.access_info	7/4/2024 5:50 PM	ACCESS_INFO File	1 KB
.compiled_spec	7/4/2024 1:12 PM	COMPILED_SPEC ...	1 KB
.hostname	7/4/2024 1:12 PM	HOSTNAME File	1 KB
.pid	7/4/2024 3:39 PM	PID File	0 KB
.view	7/4/2024 1:12 PM	VIEW File	1 KB
config_spec	7/4/2024 1:12 PM	File	1 KB
groups.sd	7/4/2024 1:12 PM	SD File	1 KB
identity.sd	7/4/2024 1:12 PM	SD File	1 KB
readme.txt	7/4/2024 1:12 PM	Text Document	1 KB
view_db.state	7/4/2024 5:50 PM	STATE File	1 KB

View directory

The main components of the view storage directory are:

View database.



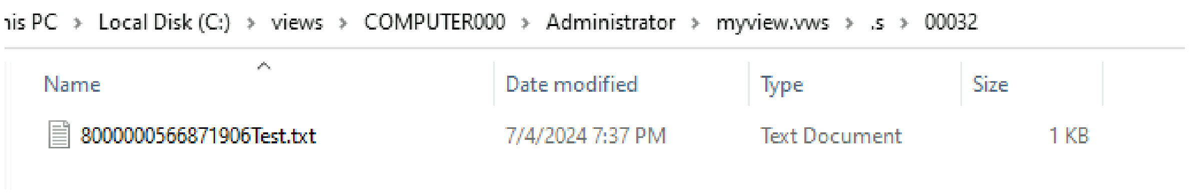
```
is PC > Local Disk (C:) > views > COMPUTER000 > Administrator > myview.vws > db

Name                        Date modified        Type            Size
view_db.d01                 7/4/2024 5:50 PM     D01 File        4 KB
view_db.d02                 7/4/2024 5:50 PM     D02 File        2 KB
view_db.dbd                 4/21/2017 9:16 AM    DBD File        8 KB
view_db.k01                 7/4/2024 5:50 PM     K01 File        2 KB
view_db_schema_version      7/4/2024 1:12 PM     File            1 KB
vista.log                   7/4/2024 3:39 PM     Text Document   4 KB
vista.taf                   7/4/2024 5:50 PM     TAF File        6 KB
```

Viewdb

- The db subdirectory contains the binary files managed by an embedded DBMS. he 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.
- The role of the files in the db subdirectory is analogous to those in the VOB db subdirectory

Private storage area

- The `.s` subdirectory is the top level of a directory tree.
- In a dynamic view, it contains all view-private objects comprising checked-out versions of file elements, unshared and nonshareable 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.
- The image below shows the `.s` contents when the element `Test.txt` is checked out


his PC > Local Disk (C:) > views > COMPUTER000 > Administrator > myview.vws > .s > 00032

Name	Date modified	Type	Size
8000000566871906Test.txt	7/4/2024 7:37 PM	Text Document	1 KB

Viewcheckout

Administrative directory.

- The admin directory contains data on disk space used by the view.
- A job that the ClearCase scheduler runs by default generates this data periodically.

View files

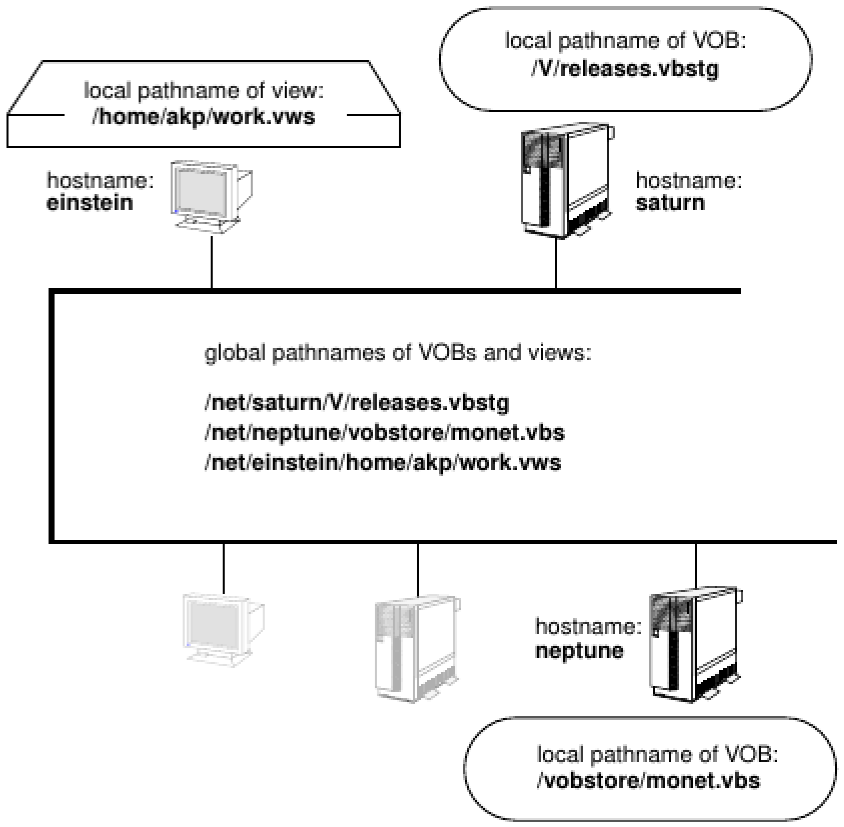
- The identity.sd and groups.sd files serve the same purpose as in the VOB directory. The .access_info file also is used in this function.
- The config_spec file contains the editable version of that View's config specs
- The .compiled_spec file contains a cached compiled version of the config specs making it faster to apply these rules when accessing elements in the view
- The .pid file serves the same function for the View process that the corresponding file in the VOB directory does for the VOB process.
- The .view file uniquely identifies the view and contains the view's UUID (Universally Unique Identifier) and other metadata that uniquely distinguish this view from all other views.

```
C:\views\COMPUTER000\Administrator\myview.vws>type .view
COMPUTER000:C:\views\COMPUTER000\Administrator\myview.vws
a1206f97.0ed04829.ae1c.bc:c3:1e:27:6e:00
COMPUTER000
```

Administering Regions

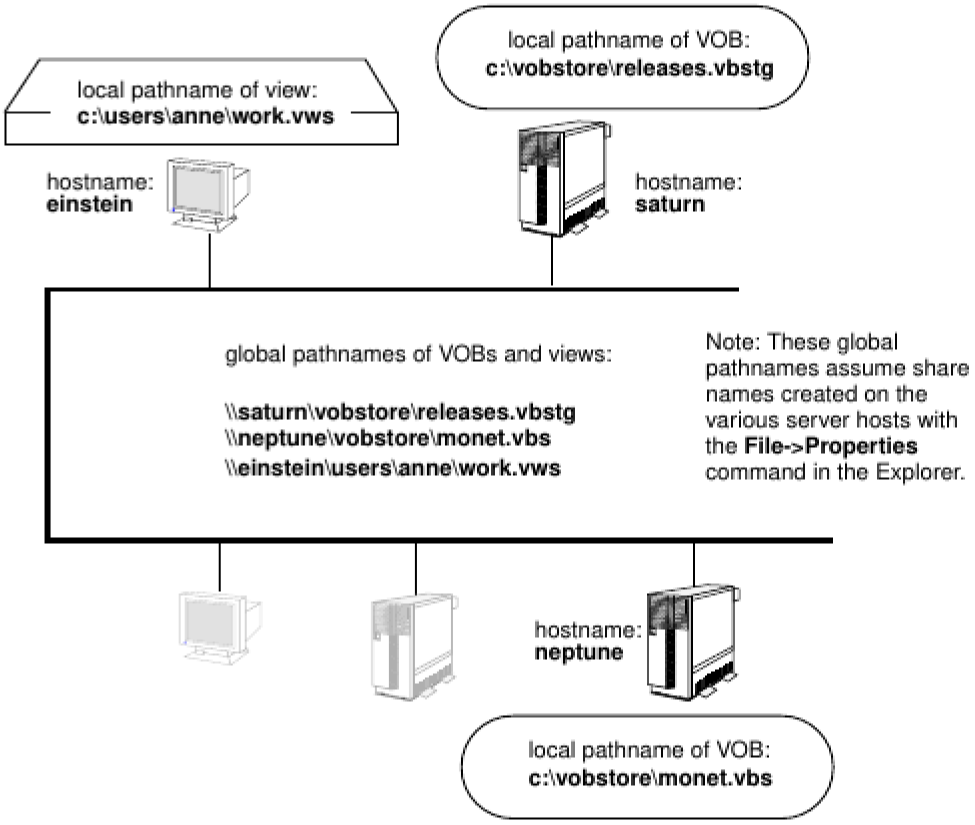
Ideally, your network's VOB and view storage directories are accessible at the same pathnames throughout the network. The diagram below shows a simple network in which global naming has been achieved.

For VOBs



Regions

And for views


local pathname of view:
c:\users\anne\work.vws
hostname:
einstein
local pathname of VOB:
c:\vobstore\releases.vbstg
hostname:
saturn
global pathnames of VOBs and views:
\\saturn\vobstore\releases.vbstg
\\neptune\vobstore\monet.vbs
\\einstein\users\anne\work.vws
Note: These global pathnames assume share names created on the various server hosts with the File->Properties command in the Explorer.
hostname:
neptune
local pathname of VOB:
c:\vobstore\monet.vbs

Views

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

1. Each ClearCase host must belong to a single network region.
2. 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.
3. Developers access VOBs and views through their VOB-tags and view-tags.
4. All hosts in a given network region use the same tags.
5. Each VOB must have a tag in the region of the host on which the VOB storage directory resides.

6. Each view must have a tag in the region of the host on which the view storage directory resides.
7. VOB or view can have tags in other regions as well.

MultiSite

Introduction

- A VOB has a single set of data containers, or storage pools, and a single database as seen in the VOB Structure section.
- With MultiSite, some or all VOBs are replicated.
- A replicated VOB is one that has been "copied" to multiple other sites
- Each site hosts a copy of the VOB, called a VOB replica
- Collectively, the set of replicas of a VOB is called a VOB family.
- Each replica includes a full set of data containers and a complete copy of the VOB database.
- Each replica functions like a regular VOB; however, there are some coordination issues that administrators must consider.
- When working with multiple copies of data, the CAP theorem applies
  - Immediate availability means that we can read new or changed data as soon as it is modified
  - Complete consistency means that the data is the same in every copy
  - The CAP theorem states that you can never have both immediate availability and complete consistency with distributed data, instead, you have to settle for some compromise between the two

VOB Replicas

- Each replica has a replica name in addition to a VOB tag.
- Both the replica name and the VOB tag are needed when creating the replica.

- For each replica, the VOB database contains a replica object that records the name of the replica.
    - The VOB database also tracks the location of each replica by host name.
    - This tracking enables identification of specific replicas at other sites with short, mnemonic identifiers without needing to know their exact locations.
- The replicas are not necessarily exact copies of each other.
- Most, but not all, of the information stored in a VOB is replicated.
    - All changes that create new data, remove old data, and move or rename existing data are propagated among the replicas in the VOB family.
- However, information stored in views accessing the VOB is not propagated.
    - For example, a replica update includes the fact that an element has been checked out, because the checkout is recorded in the VOB; but the update does not include the contents of the checked-out version.

Non-Replicated Information

Not all information is replicated across replicas.

Any information that is site-specific, usually concerning configuration in local environment is not replicated since it would be of little use. Such as:

1. Different hosts at a site may have different user spaces defined by the local password and group databases.
2. Disk configurations and capacities may vary which may require VOB storage pools to be configured independently at each replica host.
3. Different sites may have different development policies and can use site-specific scripts to enforce them.

As well, the following is not replicated

1. Trigger type objects because triggers are usually used to implement local policies, and trigger type definitions often include pathnames that do not exist at other sites.

2. Individual "attached" triggers.
3. Temporary locks, specifically those created without the `–obsolete` option.
4. Contents of checked-out versions.
5. Mastership request settings
6. Changes to text mode property. A new replica has the same text mode property as its parent replica, but later changes are not propagated. (Text mode means UNIX or Windows EOL characters)
7. Settings for synchronous request for mastership (SRFM), atomic checkins, and the minimum client feature level

Replicated Information

The following is replicated

1. Elements, branches, versions including derived object versions.
2. Most kinds of type objects.
3. Metadata annotations: version labels, attributes, hyperlinks, including merge arrows and hyperlinks to administrative VOBs
4. UCM objects: activities, baselines, components, folders, projects, streams
5. Permanent locks created with the `–obsolete` option.
6. Checkout records of elements and changes in checked-out directories.
7. Event records.
8. Setting for evil twin detection and prevention

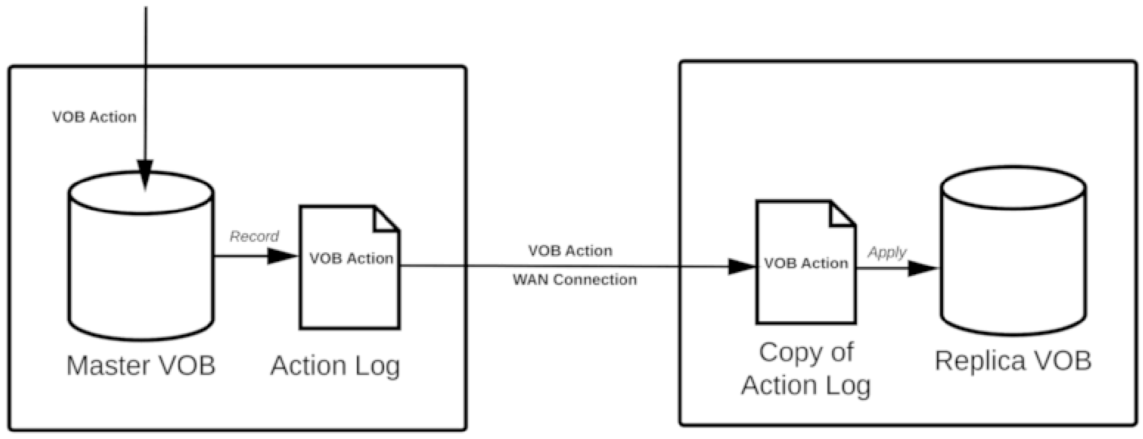
Replication Process

Because information is modified concurrently at different replica sites, the contents of each replica in a family start to diverge. In the absence of immediate updates propagating across all replicas as they occur, the contents of a specific replica will probably never be identical to the contents of any other replica.

To keep the replicas from getting too divergent, each replica sends updates to one or more other replicas. Updating a VOB replica changes both its database and its storage pools to reflect the development activity that has taken place in one or more other replicas.

Information is exported from a replica in packets.

- A logical packet includes all the information required to either:
  - Create a new replica using a *replica-creation* packet
  - Or to update one or more existing replicas using an *update* packet
- For flexibility and to accommodate limitations of data-transport facilities, each logical packet can be created as a set of physical packets.
- A logical packet is created with a `mkreplica` or `syncreplica` command used with the `–export` option
- The packet is sent to a replica
- The receiving replica processes the packet using the `mkreplica` or `syncreplica` command invoked with the `–import` option.
- The changes that were sent are added to the database and storage pools, user database and schema repository of the importing replica.
  - Updating a user database replica may change both its database and its schema repository to reflect the activity that has taken place in one or more other replicas.
- If the logical packet includes several physical packets, the import commands always process the physical packets in the correct order.
- No error occurs if the same packet is imported two or more times at a replica, unless the imports occur simultaneously.



Multi

A synchronization strategy should be developed for each family to accommodate its particular use patterns, the organization's needs, and the level of connectivity among the host machines.

- For one family, replicas might be updated every hour using a high-speed network
- For another family, updates might only once or twice a month, using electronic mail or disk files as the delivery mechanism.

Mastership

As individual replicas are modified, they increasingly diverge from each other. This would lead to large scale and possibly irreparable data corruption.

For example:

- Assume that version 17 of `util.c` is created on the main branch in three replicas at the same time.
- Which is the real version 17?
- This may depend on your perspective "Ours is the true version 17"
- If a record with the same ID is created at the same time in two replicas, which is the "real" record?

In distributed data systems, we often refer to a "system of record" or "master record" which is taken to be the "single source of truth." The term for this concept in MultiSite is "Mastership."

- CC objects are assigned a master replica or master.
- The initial master of an object is the replica where the object is created
- Mastership can be changed subsequently
- In general, an object can be modified or deleted only at its master replica.
- Replicas, VOBs, and most objects in a VOB have a master replica.
- For example, each branch type has a master replica.
  - By default, you can create or modify a branch only if your current replica masters the branch type.
  - In this example, the command fails because the current replica does not master the main branch type:
  -

```
cleartool checkout -c "add new feature" v3.0plan.doc
cleartool: Error: Unable to perform operation "checkout" in
replica
"sanfran_hub" of VOB "/vobs/doc".
cleartool: Error: Master replica of branch "/main" is
"boston_hub".
cleartool: Error: Unable to check out "v3.0plan.doc".
```

Some conflicts are unavoidable.

- For example, objects with the same name can be created at two or more VOB replicas during the same time period between synchronizations
- Or a new user named jsmith can be created at two or more sites during the same time period between synchronizations.

- Conflicts can be minimized by establishing naming conventions for objects and CC use policies
- if a conflict does occur, it is handled during the import of an update packet.

Serial and Parallel Development

In the parallel development model, mastership of a branch is given to a specific replica. For example:

- The Paris team is doing version 3.0 development using the Paris replica of the project VOB.
- Mastership of the `dev` branch resides in the Paris replica
- The New York team is doing support for the production version 2.7 using their VOB replica
- Mastership of the `rel2.7` branch resides in the New York replica

Integration of parallel development in replica follows the same general process as merging development from different branches in a single VOB.

- Changes in the `dev` and `rel2.7` would not overlap in either a single VOB between replica VOBs which means that we can still merge as usual
- This depends in both cases on having the correct isolated development and merge policies and procedures in place.

In Serial development, we cannot isolate branches in different VOBs; instead, we need to work on the same branch in two different replicas. This means that mastership of the branch has to be passed back and forth to ensure the work is not going to introduce corruption. In simple terms, only one replica at a time can make changes to a branch.

To support serial development, there are two models for changing mastership:

1. Push Model - The developer who needs to work on a branch asks the administrator of the master replica to transfer mastership of the branch and send an update packet containing the change.

2. Pull Model - The developer who needs to work on a branch requests mastership of the branch.

    - This model is not enabled by default, and it requires the MultiSite administrator to enable requests and authorize developers to request mastership.
    - However, after the setup is complete, the administrator does not need to be involved in the mastership request process.

There are two ways to use requests for mastership:

1. If a developer cannot merge versions of the element, they must request mastership. After the current replica receives mastership, the developer can perform a reserved checkout and do the merge.
2. If the developer can merge versions of the element, they can perform a nonmastered checkout of the element and do their work. At any time, they can request mastership and, when received, the work can be merged and checked in.

Mastered Checkout

A mastered checkout occurs when the checkout operation is performed on the replica that holds the master copy of the element. This is the same as a regular checkout in a non-replica VOB and is characterized by:

1. *Direct Modification* Changes are made directly to the master copy of the element.
2. *Synchronization* There is no need for additional synchronization steps specific to the checkout process itself since the master copy is being updated.
3. *Conflict Management* Since the element is mastered at the current site, there are fewer chances of conflicts arising due to concurrent changes at different sites.
4. *Performance* Operations may be faster because no remote communication is necessary for modifying the master copy.

```
cleartool checkout -c "Checking out a mastered element" filename
```

Non-mastered Checkout

A non-mastered or unmastered checkout occurs when the checkout operation is performed on a replica that does not hold the master copy of the element. This is characterized byL

1. *Local Copy* A non-mastered checkout creates a local copy of the element for modifications at the current site.
2. *Synchronization Required* After making changes, synchronization with the master replica is required to propagate changes and resolve any potential conflicts.
3. *Conflict Management* There is a higher risk of conflicts, as changes need to be merged with the master copy maintained at another site.
4. *Performance Considerations* Operations may involve additional overhead due to the need to synchronize changes with the master replica.

VOB objects and VOB replica objects

Each replica is a VOB, but the VOB object and VOB-replica object are different objects in the VOB database. Specifically:

- *VOB object* The database has a single VOB object. This object's UUID is listed as the VOB family uuid in a `lsvob –long` listing.
- *VOB-replica object (or replica object)* The database has a VOB-replica object for each of the VOB's replicas. This object's UUID is listed as the Vob replica uuid in a `lsvob –long` listing.

```
cleartool lsvob -long /vobs/dev
Tag: /vobs/dev
...
Vob family uuid: 87f6265b.72d911d4.a5cd.00:01:80:c0:4b:e7
Vob replica uuid: 87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7
Vob registry attributes: replicated
```

Use describe vob: to list details about the VOB object; use `describe replica:` to list details about the VOB-replica object (the replica). For example:

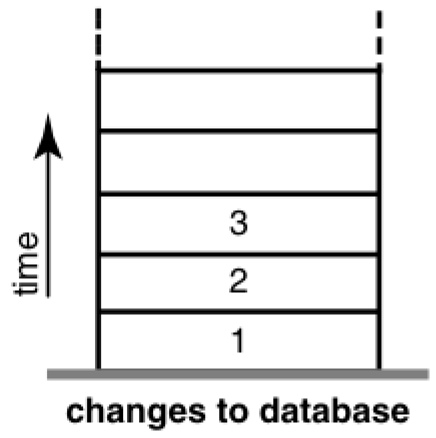
```
cleartool describe vob:/vobs/dev replica:boston_hub@/vobs/dev
versioned object base "/vobs/doc"
  created 15-Aug-01.14:19:48 by susan.user
  "Main source VOB for development."
  master replica: boston_hub@/vobs/dev
  replica name: boston_hub
  VOB family feature level: 3
  VOB storage host:pathname "goldengate:/vobstg/dev.vbs"
  VOB storage global pathname "/net/goldengate/vobstg/dev.vbs"
 ...
replica "boston_hub"
  created 15-Aug-01.14:19:48 by susan.user
  replica type: unfiltered
  master replica: boston_hub@/vobs/dev
  request for mastership: enabled
...
```

Operation Log

For a replica family, changes are tracked for each replica in an `oplog`.

- An oplog entry includes the identity of the replica where the operation originated.
- The history of a replica family can be viewed as several stacks of oplog entries.
- Each stack is represented by a linear sequence of oplog IDs for the operations that originate in that replica.

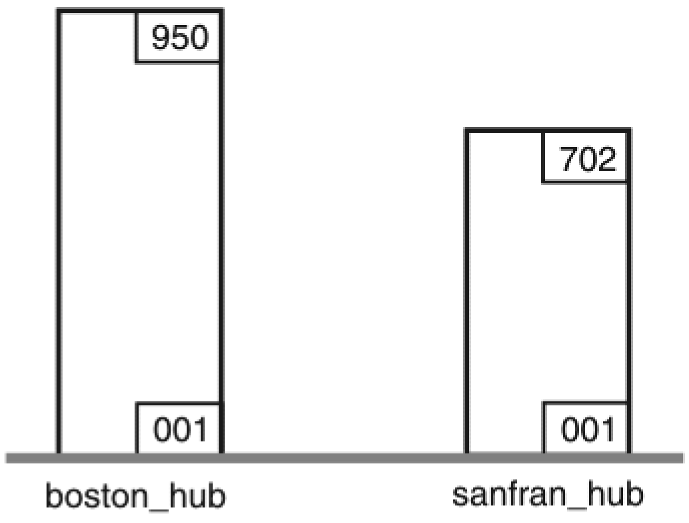
The history of an unreplicated VOB database is a linear sequence of operations



Chg vob

The state of a family is a has to account for multiple histories of operations. For example, in the image below:

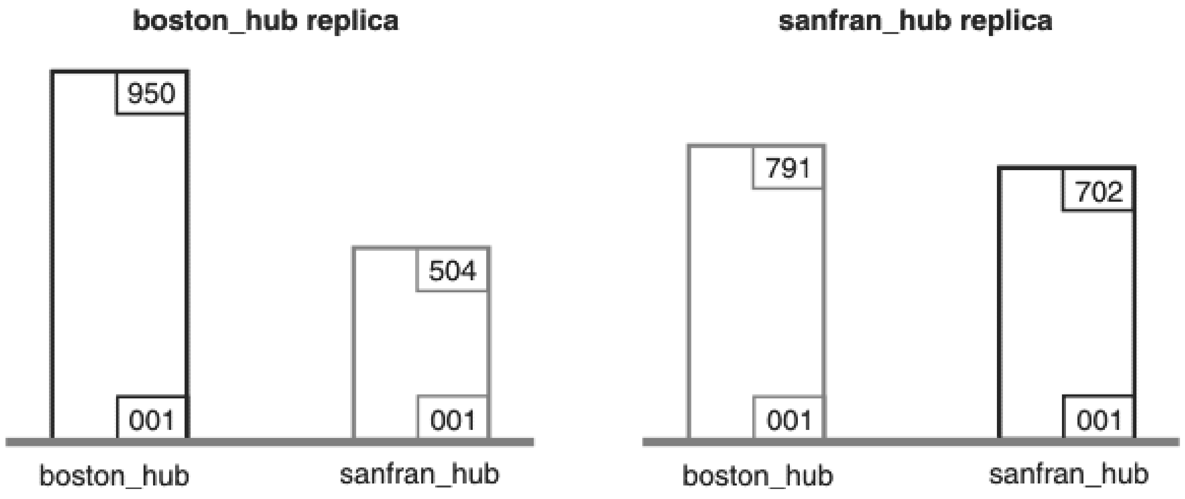
- Operations with oplog IDs 1–950 have occurred at replica boston_hub.
- Operations 1–702 have occurred at replica sanfran_hub.



Lsf char

- A replica has accurate data only about its own operations.
- Until it receives update packets, its information about other replicas is out of date.

- For example, replica boston_hub records 950 local operations, but has received update packets for only 504 sanfran_hub operations.
- Similarly, replica sanfran_hub records 702 local operations, but has received update packets for only 791 boston_hub operations.
-

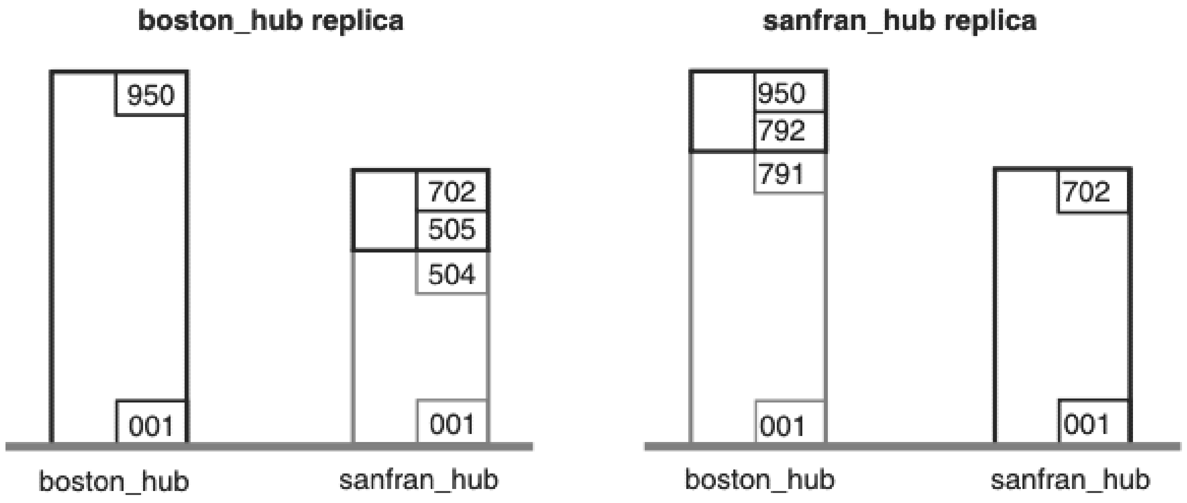


Lsf rep1

Picturing a replica family as a set of oplog stacks, shown above, makes it easy to understand the synchronization process.

For example, an update packet sent from replica boston_hub to replica sanfran_hub consists of increments to the stack for replica boston_hub (operations 792–950).

- The diagram below shows the two increments.
- Because sanfran_hub knows its own state, it needs updates only for operations originating at other replicas.



Lsf rep2

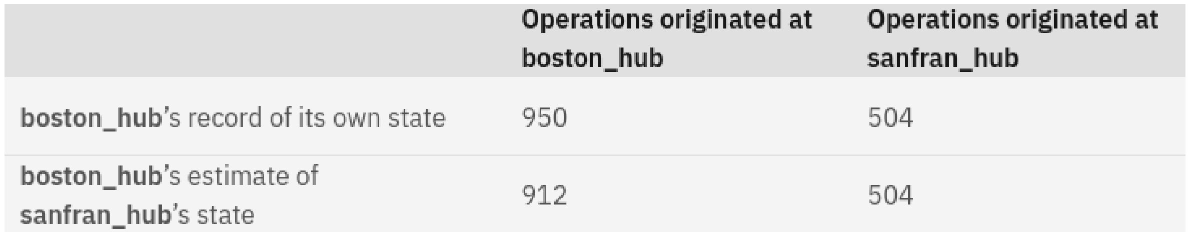
Epoch numbers

An epoch number is the total number of operations that originated at a particular replica.

In the previous diagrams, the epoch number for boston_hub is 950. MultiSite synchronization scheme attempts to minimize the amount of data transmitted among replicas.

Each replica keeps track of these epoch numbers:

- Changes made in the current replica. The number of operations that originated at the current replica.
- Changes in sibling replicas that have been imported to the current replica. When synchronized replica writes an operation from an update packet to the current replica, it increments the epoch number that records the number of operations originating at the sibling replica that have been imported at the current replica.
- Estimates of the states of other replicas. For each other replica, an estimate of its own changes and other replicas' changes. The current replica keeps track of the operations it has sent to other replicas, and assumes that these operations are imported successfully.



	Operations originated at boston_hub	Operations originated at sanfran_hub
boston_hub's record of its own state	950	504
boston_hub's estimate of **sanfran_hub**'s state	912	504

Matrix

Each replica maintains a matrix like the one above, revising its rows as work occurs locally and as it exchanges update packets with other replicas:

- When work occurs in the boston_hub replica, its own epoch number is incremented.
- When the boston_hub replica receives an update from sanfran_hub, it revises its own row (boston_hub) and the sanfran_hub row in its epoch number matrix.
- When the boston_hub replica generates an update packet to be sent to sanfran_hub, it revises the sanfran_hub row in its epoch number matrix.

A `syncreplica –export` command updates epoch numbers immediately.

- It does not wait for acknowledgment from the importing replica that the packet has been received and applied correctly.
- During normal MultiSite processing, manual intervention is not necessary to maintain the accuracy of the epoch number matrices for the various replicas. However, failure to apply a packet may require manual intervention.

The contents of this matrix are reported by the `lsepoch` command at the boston_hub replica:

```
multitool lsepoch
For VOB replica "/vobs/dev":
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950     (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504     (sanfran_hub)
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=912     (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504     (sanfran_hub)
```

```
multiutil lsepoch -clan telecomm -site boston_hub -family PRODA -
user bostonadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at site
'boston_hub' (@minuteman):
boston_hub: 950
sanfran_hub: 504
Multiutil: Estimates of the epochs from each site replayed at site
'sanfran_hub' (@goldengate):
boston_hub: 912
sanfran_hub: 504
```

A syncreplica –export command entered at boston_hub uses this matrix as follows to generate an update destined for sanfran_hub:

1. At the boston_hub replica, the number of local operations is 950 (the number in the upper left corner of matrix), and the estimate is that the sanfran_hub replica has imported all operations through oplog ID 912 (the number in lower left corner).
2. The update packet that the boston_hub replica sends to the sanfran_hub replica includes boston_hub oplog entries 913-950. After the Boston administrator invokes syncreplica –export, the sanfran_hub row is updated:

```
multitool lsepoch
For VOB replica "/vobs/dev":
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950     (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504     (sanfran_hub)
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950     (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504     (sanfran_hub)


multiutil lsepoch -clan telecomm -site boston_hub -family PRODA -
user lexadmin -password secret
```

```
Multiutil: Estimates of the epochs from each site replayed at site
'boston_hub' (@minuteman):
boston_hub: 950
sanfran_hub: 504
Multiutil: Estimates of the epochs from each site replayed at site
'sanfran_hub' (@goldengate):
boston_hub: 950
sanfran_hub: 504
```

Installation

MultiSite is installed as an optional product component during the CC

- MultiSite must be installed on all VOB server hosts where replicated VOBs will reside; replica creation must occur on the host where the replica resides.
- You do not need to install MultiSite on the user's computer to manage mastership because the MultiSite object mastership commands are available in cleartool.
- However, installing MultiSite on the user's computer provides convenient access to other MultiSite commands.
- MultiSite does not need to be installed pn server hosts that will not host replicated VOBs.

Each site needs a synchronization server to handle packet transport.

- This host must have the Rational Shipping Server installed on it.
- Each site also needs a MultiSite administrative host, which must have the Rational ClearQuest MultiSite Administration Tools (multiutil) installed on it. On this host, you run multiutil commands to synchronize and manage replicas.

A MultiSite licence is required to access an object in a replica by a MultiSite command or GUI, or by a standard operating system command.

Creating VOB replicas

VOB replication is a three-phase procedure to create a new replica: export, transport, and import. To create a new replica:

Export phase

The export phase of the replica creation process occurs at the original site.

- This phase consists of running the export form of the mkreplica command,
- Taking a backup of the original VOB including the info that the VOB has now been replicated
- Verifying that the replica-related changes occurred.

These steps take place at the original site.

- Enter the export form of the mkreplica command on a single line. In this example, the administrator uses the –fship option to send the packet immediately.

```
MINUTEMAN% multitool mkreplica -export -workdir /tmp/ms_wkdir
-fship goldengate:sanfran_hub@/vobs/dev
Enabling replication in VOB.
Comments for "sanfran_hub":
First time replication for dev VOB
Creating new replica, sanfran_hub, on host goldengate
.

Generating replica creation packet
/opt/rational/clearcase/shipping/ms_ship/
outgoing/repl_boston_hub_1
6-Aug-03.09.49.36_14075_1
- shipping order file is
 /var/adm/rational/clearcase/shipping/ms_ship/outgoing/
sh_o_repl_boston_hub_16-Aug-03.09.49.36_14075_1
Dumping database...
...

Dumper done.
```

```
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/
outgoing/repl_boston_hub_1
6-Aug-03.09.49.36_14075_1
```

- Back up the original VOB. This backup records the fact that the VOB is replicated. If it is necessary to restore a VOB replica from a backup copy that was made before the VOB was replicated, the MultiSite replica restoration procedure fails.
- Verify the replica-related changes. The `mkreplica` command creates a new replica object in the VOB database. This object is similar to the VOB object that represents the entire VOB in the database, and its properties are listed by the `lsreplica` command.

```
% multitool lsreplica -invob /vobs/dev
For VOB replica "/vobs/dev":
15-Aug.14:19   susan       replica "boston_hub"
16-Aug.09:49   susan       replica "sanfran_hub"
```

Transport Phase

In the transport phase, which takes place at the original site, the replica-creation packet is sent to the new site. This process differs, depending on how `mkreplica` command is executed during the export phase:

- If the `–fship` option of the `mkreplica` command is specified during the Export phase, the packet was shipped automatically at that point and no further action is necessary here
- If the `–ship` option of the `mkreplica` command is specified during the Export phase, then the `shipping_server` command must be used to ship the replica-creation packet to the new site, as shown below.
- To manually ship a replica-creation packet stored in the shipping server, use the shipping_server command:

```
/opt/rational/clearcase/etc/shipping_server -poll
```

Import Phase

The import phase of the replica creation process occurs at the new replica's site. It consists of:

- Verifying the packet's arrival using the `lspacket` command
- Creating the replica using the import form of the `mkreplica` command
- Deleting the replica creation packet if necessary
- Setting the replica properties such as identities and permissions, self-mastership, and feature level.

Verify the packet's arrival by running `lspacket` on the receiving host. By default, `lspacket` searches all the MultiSite storage bays for packets. For example, if host `goldengate` is the receiving host:

```
GOLDENGATE% multitool lspacket
Packet is: /opt/rational/clearcase/shipping/ms_ship/
incoming/repl_boston_hub_1
6-Aug-03.09.49.36_14075_1
Packet type: Replica Creation
VOB family identifier is: 12a3456b.78c901d2.e3ab.45:67:89:c0:1d:e2
Comment supplied at packet creation is:
Packet intended for the following targets:
sanfran_hub
The packet sequence number is 1
```

Enter the import form of the replica-creation command.

- This replica is permissions-preserving, so the user who executes this command becomes the owner of the new VOB replica and all elements in it. This user’s primary group is the group for all elements. Typically, administration is easier if this user is not the root user or a member of the CC administrators group.

- The `Should I create this replica?` prompt can be bypased during replica creation by specifying the `–vreplica` option with the new replica’s name. This example does not specify that option.
- If a permissions-preserving replica is being created, the prompt during replica creation can be bypassed by specifying the `–nprompt` option with the `–perms_preserve` option.
- The pathname of the incoming packet as listed by the `lspacket` command must be specified