ClearCase Reference Manual

Release 4.1 and later

UNIX Edition

Rational®
the e-development company™

800-023815-000

ClearCase Reference Manual
Document Number 800-023815-000 November 2000
Rational Software Corporation 20 Maguire Road Lexington, Massachusetts 02421

IMPORTANT NOTICE

Technical Acknowledgments

This software and documentation is based in part on BSD Networking Software Release 2, licensed from the Regents of the University of California. We acknowledge the role of the Computer Systems Research Group and the Electrical Engineering and Computer Sciences Department of the University of California at Berkeley and the Other Contributors in its development.

This software and documentation is based in part on software written by Victor A. Abell while at Purdue University. We acknowledge his role in its development.

This product includes software developed by Greg Stein <gstein@lyra.org> for use in the mod_dav module for Apache (http://www.webdav.org/mod_dav/).

Rational®
the e-development company™

Contents

Figures

Tables

Preface

ClearCase® is a comprehensive software configuration management system. It manages multiple variants of evolving software systems, tracks which versions were used in software builds, performs builds of individual programs or entire releases according to user-defined version specifications, and enforces site-specific development policies.

ClearCase LT offers capabilities like those of ClearCase, but for the smaller software development group.

ClearCase Attache™ (abbreviated to "Attache" in this manual) provides a ClearCase client solution for Microsoft® Windows® users. For more information, see the *ClearCase Attache Manual*.

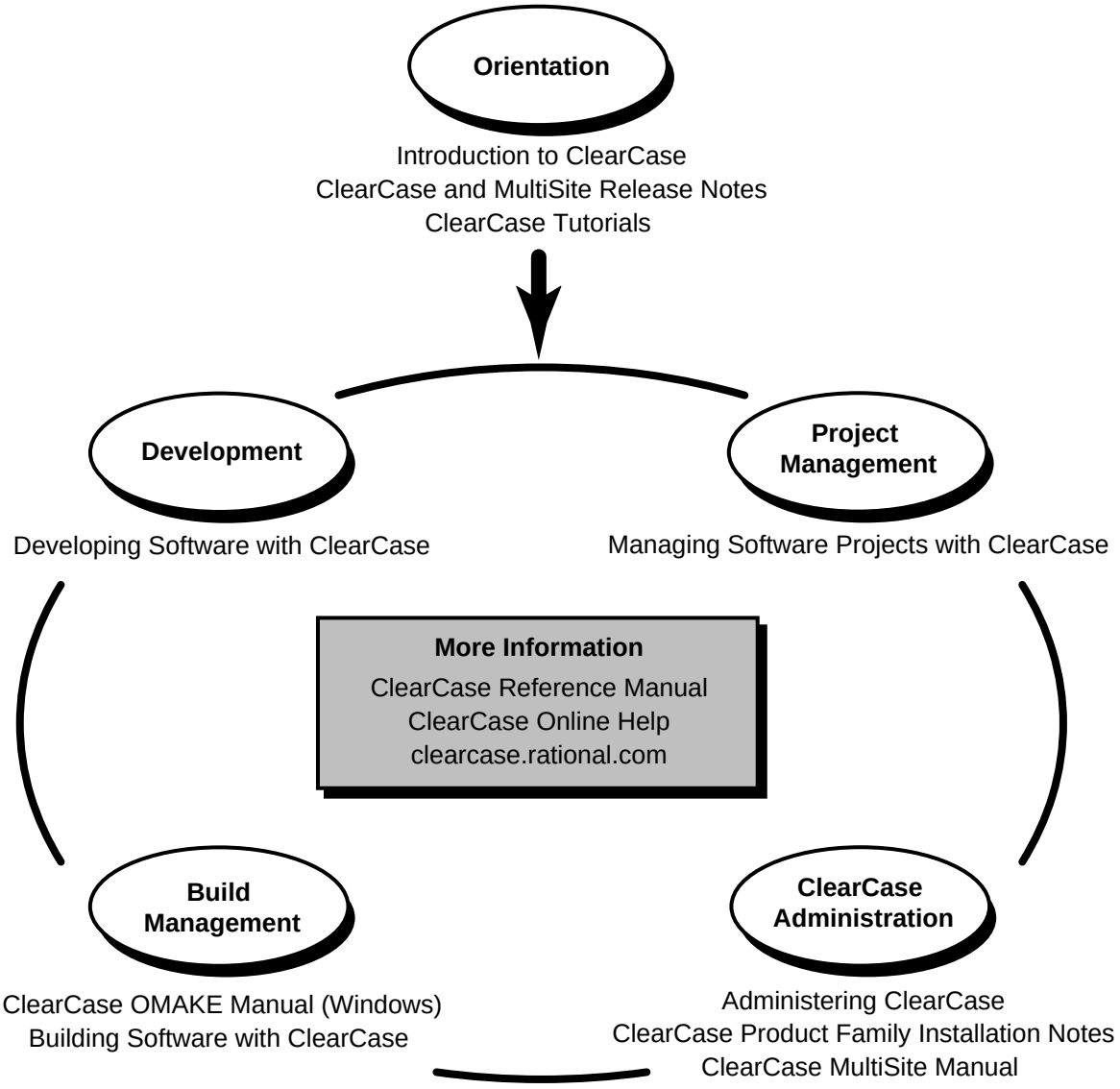
ClearCase MultiSite® (abbreviated to "MultiSite" in this manual) is a layered product option for ClearCase. It supports parallel software development and software reuse across project teams that are distributed geographically.

About This Manual

This manual includes detailed reference information for ClearCase, ClearCase LT, Attache, and MultiSite. It describes command syntax and use, and is not intended to be a learning tool. This manual assumes you have already learned about these products through other means.

The reference pages are in alphabetical order in two volumes. Each reference page has an Applicability section that lists the products to which the page applies. Within each reference page, product-specific information is annotated "ClearCase only," "ClearCase LT only," and so on. In this context, the term *ClearCase* always refers only to ClearCase, not to ClearCase LT, ClearCase Attache, ClearCase MultiSite, nor to the ClearCase Product Family (CPF) in general.

ClearCase Documentation Roadmap



Typographical Conventions

This manual uses the following typographical conventions:

- *ccase-home-dir* represents the directory into which the ClearCase Product Family has been installed. By default, this directory is **/usr/atria** on UNIX and **C:\Program Files\Rational\ClearCase** on Windows.
- *attache-home-dir* represents the directory into which ClearCase Attache has been installed. By default, this directory is **C:\Program Files\Rational\Attache**, except on Windows 3.*x*, where it is **C:\RATIONAL\ATTACHE**.
- **Bold** is used for names the user can enter; for example, all command names, file names, and branch names.
- *Italic* is used for variables, document titles, glossary terms, and emphasis.
- `A monospaced font` is used for examples. Where user input needs to be distinguished from program output, **bold** is used for user input.
- Nonprinting characters are in small caps and appear as follows: **<EOF>**, **<NL>**.
- Key names and key combinations are capitalized and appear as follows: SHIFT, CTRL+G.
- [ ] Brackets enclose optional items in format and syntax descriptions.
- { } Braces enclose a list from which you must choose an item in format and syntax descriptions.
- | A vertical bar separates items in a list of choices.
- ... In a syntax description, an ellipsis indicates you can repeat the preceding item or line one or more times. Otherwise, it can indicate omitted information.

  NOTE: In certain contexts, ClearCase recognizes "**...**" within a pathname as a wildcard, similar to "*" or "?". See the **wildcards_ccase** reference page for more information.
- If a command or option name has a short form, a "medial dot" (·) character indicates the shortest legal abbreviation. For example:

  lsc·heckout

  This means that you can truncate the command name to **lsc** or any of its intermediate spellings (**lsch**, **lsche**, **lschec**, and so on).

Command Examples

Reference pages for commands have usage examples. The examples for **cleartool** subcommands and Attache commands begin with the *cmd-context* variable. This reflects the fact that the commands are invoked differently, depending on the operating context:

- Attache — *cmd-context* represents the workspace prompt. If the example looks like this:

  cmd-context **checkin –nc hello.c**

  you would enter the following at the workspace prompt:

  checkin –nc hello.c

- ClearCase in single-command mode — *cmd-context* indicates that you must type **cleartool**, then the rest of the input, at your regular command prompt. If the example looks like this:

  cmd-context **checkin –nc hello.c**

  you would enter the following at your command prompt:

  cleartool checkin –nc hello.c

- ClearCase in interactive **cleartool** mode — *cmd-context* represents the interactive **cleartool** prompt. If the example looks like this:

  cmd-context **checkin –nc hello.c**

  you would enter the following at the `cleartool>` prompt (type **cleartool** to enter interactive mode):

  checkin –nc hello.c

Online Documentation

The ClearCase Product Family (CPF) graphical interfaces include a Microsoft Windows-like help system.

There are three ways to access the online help system: the **Help** menu, the **Help** button, or the F1 key. **Help**>**Contents** provides access to the complete set of online documentation. For help on a

particular context, press F1. Use the **Help** button on various dialog boxes to get information specific to that dialog box.

CPF products also provide access to full reference pages (detailed descriptions of commands, utilities, and data structures) using the **man** command. Without any argument **man** displays the overview reference page for the command line interface. For information about using a particular command, specify the command name as an argument.

Examples:

`% cleartool man` *(display the cleartool overview page)*

`% clearguide man` *(display the man reference page)*

`attache-workspace> man checkout` *(display the Attache checkout reference page)*

CPF products provide access to syntax for individual commands. The **–help** command option displays individual subcommand syntax. For example:

```
% cleartool lsprivate –help
Usage: lsprivate [-tag view-tag] [-invob vob-selector] [-long | -short]
                 [-size] [-age] [-co] [-do] [-other]
```

Without any argument, **cleartool help** displays the syntax for all **cleartool** commands.

The **apropos** command displays command summary information and entries from the ClearCase glossary. See the **apropos** reference page for more information.

Additionally, the online tutorials provide important information on setting up a user's environment, along with a step-by-step tour through each product's most important features.

Technical Support

If you have any problems with the software or documentation, please contact Rational Technical Support via telephone, fax, or electronic mail as described below. For information regarding support hours, languages spoken, or other support information, click the **Technical Support** link on the Rational Web site at **www.rational.com**.

Your Location	Telephone	Facsimile	Electronic Mail
North America	800-433-5444 toll free or 408-863-4000 Cupertino, CA	408-863-4194 Cupertino, CA 781-676-2460 Lexington, MA	**support@rational.com**
Europe, Middle East, and Africa	+31-(0)20-4546-200 Netherlands	+31-(0)20-4546-201 Netherlands	**support@europe.rational.com**
Asia Pacific	61-2-9419-0111 Australia	61-2-9419-0123 Australia	**support@apac.rational.com**

make

Executes a make program in the current *working directory* of your workspace

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

make [*arg ...*]

DESCRIPTION

The **make** command executes a make program in the current working directory of your workspace; this directory must exist locally. On Windows 3.x, the **make** command invokes *attache-home-dir***\etc\wsmake.pif**, which can be customized to run the **make** program of your choice. On Windows NT and Windows 95, **make** looks for an environment variable named **WSMAKE**. If found, **make** uses the value of the **WSMAKE** environment variable as the name of the program to run. Otherwise, **make** runs **nmake**, which must be on your **PATH**.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

Make Program. *Default:* No arguments are passed to the **make** program.

arg ...

Optionally, passes one or more arguments to the **make** program.

EXAMPLES

- Run the **make** program in your workspace.

  make -n -v -k

SEE ALSO

attache_command_line_interface, **wshell**

man

Displays an online reference page

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command
MultiSite	multitool subcommand

SYNOPSIS

man [**–g·raphical**] [*command_name*]

DESCRIPTION

This command does not require a product license.

The **man** command displays the specified online reference page in flat-ASCII format. With the **–graphical** option, the reference page is displayed in Windows-style help format in a separate help viewer. For **cleartool** and **multitool** subcommands, Attache local commands, and hybrid commands, or if your Attache helper is running on a UNIX host, you can use any valid command abbreviation or alias. For example:

cmd-context **man lscheckout**	*(abbreviation; in Attache, valid for local or hybrid command or UNIX only)*
cmd-context **man lsch**	*(full command name)*
cmd-context **man lsco**	*(alias; in Attache, valid for local or hybrid command or UNIX only)*

With no arguments, **man** displays the product's overview reference page. In Attache, **help** is a synonym for **man**.

USE OF MANPATH

(This section does not apply to Attache.)

Reference pages are stored in subdirectories of *ccase-home-dir***/doc/man**. The **man** subcommand modifies the environment to include a **MANPATH** variable set to this directory. It then executes the UNIX **man(1)** command in a subprocess. Thus, the shell from which you invoke **cleartool** need not have **MANPATH** set.

If, however, you want to use UNIX **man** directly, without going through **cleartool**, **multitool**, or **clearguide**, be sure to include *ccase-home-dir*/**doc/man** in your **MANPATH**. For example:

setenv MANPATH /usr/catman:/usr/man:/usr/atria/doc/man

Note that with UNIX **man**, you must match the reference page file name. File names of **cleartool** subcommands have a **ct+** prefix; file names of **multitool** subcommands have a **mt+** prefix; file names of **clearguide** subcommands have a **cg+** prefix:

% **man ct+describe**	*(correct)*
% **man mt+syncreplica**	*(correct)*
% **man cg+mkactype**	*(correct)*
% **man describe**	*(incorrect)*
% **man des**	*(incorrect)*

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

(NOT IN ATTACHE) DISPLAYING THE REFERENCE PAGE IN HELP FORMAT. *Default:* Displays the reference page in flat-ASCII format.

–g·raphical
Starts a help viewer to display the reference page.

SPECIFYING THE REFERENCE PAGE. *Default:* Displays the overview reference page for the product.

command_name
The name (or abbreviation, or alias) of a **cleartool** or **multitool** subcommand, Attache local or hybrid command, or the name of any other ClearCase, ClearCase LT, or MultiSite reference page.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Display the reference page for the **mkview** command.

  cmd-context **man mkview**

- Display the overview reference page for the product.

  cmd-context **man**

SEE ALSO

attache_command_line_interface, help

merge

Merges versions of a text-file element or a directory

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and ClearCase LT only:

  merge { **–out** *output-pname* | **–to** *contrib-&-result-pname* }
  [**–g**·**raphical** [**–tin**·**y**] |
  [**–tin**·**y** | **–win**·**dow**] [**–ser**·**ial_format** | **–dif**·**f_format** | **–col**·**umns** *n*]]
  [**–bas**·**e** *pname* | **–ins**·**ert** | **–del**·**ete**] [**–nda**·**ta** | **–nar**·**rows**] [**–rep**·**lace**]
  [**–q**·**uery** |**–abo**·**rt** | **–qal**·**l**]
  [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
  [**–opt**·**ions** *pass-through-options*]
  { **–ver**·**sion** *contrib-version-selector ...* | *contrib-pname ...* }

- Attache only:

  merge { **–out** *output-pname* | **–to** *contrib-&-result-pname* }
  { **–g**·**raphical** [**–tin**·**y**] [**–nda**·**ta** | **–nar**·**rows**] [**–q**·**uery** |**–abo**·**rt** | **–qal**·**l**] |
  { **–nda**·**ta** | **–abo**·**rt** [**–nar**·**rows**] }
  [**–ser**·**ial_format** | **–dif**·**f_format** | **–col**·**umns** *n*] }
  [**–bas**·**e** *pname* | **–ins**·**ert** | **–del**·**ete**] [**–rep**·**lace**]
  [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
  [**–opt**·**ions** *pass-through-options*]
  { **–ver**·**sion** *contrib-version-selector ...* | *contrib-pname ...* }

DESCRIPTION

ClearCase and ClearCase LT only

The **merge** command calls an element-type-specific program (the merge method) to merge the contents of two or more files, or two or more directories. Typically the files are versions of the same file element. A directory merge must involve versions of the same directory element.

When used to merge directory versions in a *snapshot view*, this command also updates the directory (and subdirectories, if necessary). (See **update**.)

You can also perform a subtractive merge, which removes from a version the changes made in one or more of its predecessors.

merge uses the type manager mechanism to select a **merge** method. For details, see the **type_manager** reference page. **merge** methods are supplied only for certain element types.

Attache only

This command merges the contents of two or more files, or two or more directories. Typically the files are versions of the same file element. A directory merge must involve versions of the same directory element.

merge presumes that all files are text files, using the built-in textual **diff** and **merge**, and bypassing the type manager mechanism. Any missing file contributors are downloaded temporarily to the workspace. For a directory merge, the text file encodings of the directories are downloaded. If the merge is successful and should have a merge hyperlink created, a remote **merge –ndata** command is issued to create the hyperlink. The merged result is not uploaded to the view until a **checkin** or **put** of the result occurs. Local directories are not updated after a directory merge; you must issue **get** commands to update merged directories.

You can also perform a subtractive merge, which removes from a version the changes made in one or more of its predecessors.

FILE MERGE ALGORITHM

A merge is a straightforward extension of a file comparison. Instead of displaying the differences, the **merge** method (ClearCase and ClearCase LT) or **merge** (Attache) analyzes them (sometimes with your help) and copies sections of text to the output file:

- Sections in which there are no differences among the contributors are copied to the output file.
- For differences in which exactly *one* contributor differs from the base contributor, the **merge** method (ClearCase and ClearCase LT) or **merge** (Attache) accepts the change and copies the contributor's modified section to the output file:

```
------------[changed 3-4]----|--------[changed to 3-4 file 2]---
now is the thyme                      | now is the time
for all good men                      | for all good people
                                     -|-
*** Automatic: Applying CHANGE from file 2 [lines 3-4]
============
```

(In ClearCase and ClearCase LT, the **–qall** option turns off automatic acceptance of this kind of change.)

- (ClearCase and ClearCase LT only) For differences in which two or more contributors differ from the base contributor, the **merge** method detects the conflict, and prompts you to resolve it. It displays all contributor differences, and allows you to accept or reject each one for inclusion in the output file.

```
               [changed 10]        |          [changed to 10 file 2]---
cent                               | sent
                                  -|-
               [changed 10]        |          [changed to 10 file 3]---
cent                               | scent
                                   |-
Do you want the CHANGE made in file 2? [yes] no
Do you want the CHANGE made in file 3? [yes] yes
Applying CHANGE from file 3 [line 10]
============
```

  Be sure to verify that the changes you accept produce consistent merged output. For example, after performing a merge involving file **util.c**, you can compare files **util.c.contrib** (which contains its previous contents) and the new **util.c** (which contains the merge output).

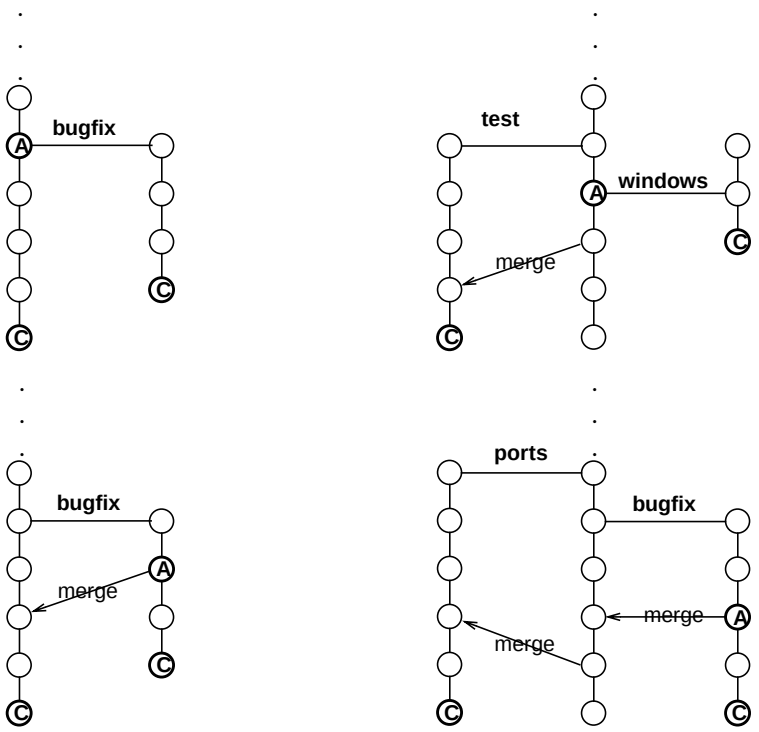
- (Attache only) Be sure to verify that the changes you accept produce consistent merged output. For example, if you perform a merge involving a file **util.c** immediately after you check out the file, you can use the **diff** command with the **–pred** option to compare the new **util.c** in the workspace (containing the merge output) with the original **util.c** in the view. However, if you've made changes to the file since you checked it out, you can make a copy of the modified local file in your workspace before performing the merge. After the merge, you can compare the new **util.c** with the premerge local copy.

Determination of the Base Contributor

If all the contributors are versions of the same element, **merge** determines the base contributor automatically. It examines the element's "merge-enhanced" version tree, which is the directed graph consisting of the actual version tree along with all the merge arrows created by previous merge operations. This examination reveals the relationships among versions from the standpoint of their contents (which versions contributed their data to me?), rather than from the standpoint of their creation order (which versions were created before me?). **merge** selects the closest common ancestor in this enhanced version tree as the base contributor.

If no merges have been performed in the element, the actual common ancestor (**A**) of the contributors (**C**) in the version tree is selected to be the base contributor. Figure 8 illustrates some common cases.

Figure 8 Determination of the Base Contributor for a Merge



If the contributors are not all versions of the same element, there is no base contributor. This means that you must resolve all discrepancies among the contributors. In ClearCase and ClearCase LT only, the **–qall** option is enabled automatically.

Recording of Merge Arrows

Under certain circumstances, **cleartool** or Attache records the merge by creating one or more merge arrows (hyperlinks of type **Merge**):

- All contributor files must be versions of the same file element.
- One of the contributors must be a checked-out version, and you must specify this version as the target to be overwritten with the merge output (**–to** option). (Alternatively, you can use the **–ndata** option to create merge arrows without performing a merge; in this case, you do not need to check out any of the contributors.)
- You must not use the **–narrows** option, which suppresses creation of merge arrows.
- You must not use any of these options: **–insert**, **–delete**, **–base**.

If all these conditions hold, **cleartool** or Attache draws an arrow from each contributor version (except the base contributor) to the target version.

In ClearCase and ClearCase LT, you can view merge arrows with the **xclearcase** Version Tree Browser by clicking **Versions** > **Show version tree**.

The **find** and **lsvtree –merge** commands can locate versions with **Merge** hyperlinks. The **describe** command lists all of a version's hyperlinks, including merge arrows:

```
cmd-context  describe util.c@@/main/3
version "util.c@@/main/3"
.
.
.
 Hyperlinks:
 Merge@278@/vob_3 /vob_3/src/util.c@@/main/rel2_bugfix/1
 -> /vob_3/src/util.c@@/main/3
```

DIRECTORY MERGE ALGORITHM

Each version of a ClearCase or ClearCase LT directory element contains the names of certain file elements, directory elements, and VOB symbolic links. **merge** can process two or more versions of the same directory element, producing a directory version that reflects the contents of all the contributors. The algorithm is similar to that for a file merge: **merge** compares a base contributor (common ancestor) version with each other contributor, producing a set of differences. It applies these differences to the base contributor as automatically as possible, prompting for user interaction only when two or more of the contributors are in conflict. (See the **diff** reference page for more on this algorithm.)

One of the directory versions—the merge target, specified with the **–to** option—must be checked out. (Typically, it is the version in your view.) **merge** updates the checked-out directory by adding, removing, and changing names of elements and/or links.

NOTE: In ClearCase and ClearCase LT, a directory merge does not leave behind a **.contrib** file, with the premerge contents of the target version.

We recommend that you use this procedure when merging directories:

1. Make sure that all contributor versions of the directory are checked in.
2. Check out the target version of the directory.
3. Perform the directory merge immediately, without making any other changes to the checked-out version.

This procedure makes it easy to determine exactly what the merge accomplished: enter a **diff –predecessor** command on the checked-out version, which has just been updated by **merge**.

Using ln and rmname to Implement a Merge

ClearCase, ClearCase LT, and Attache implement directory merges using VOB hard links. You can use the **ln** and **rmname** commands to perform full or partial merges manually. See the **ln** and **rmname** reference pages for details.

COMMON SCENARIOS

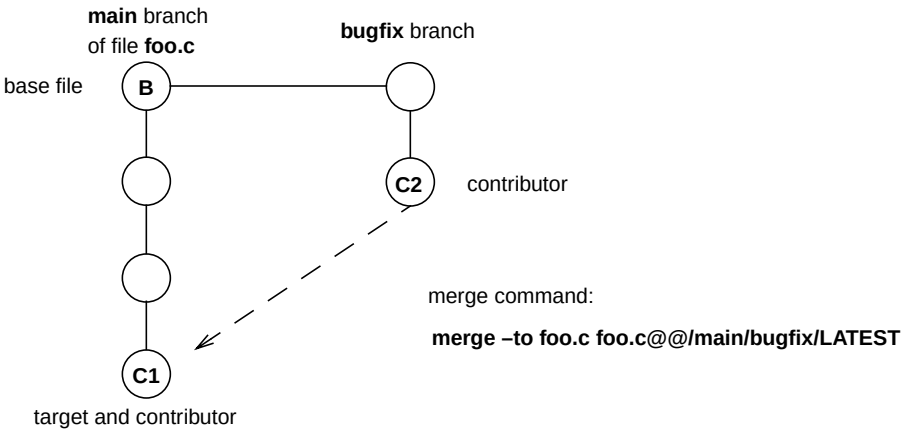
This section presents common scenarios for performing merges.

Case 1: Merging from a Branch

A common use of this command is to combine the changes made on a subbranch (for example, a **bugfix** branch) of a file element with changes made on the **main** branch. Figure 9 shows such a merge.

The target is **C1**, the checked-out version on the **main** branch, and the other contributor is **C2**, the latest version on a **bugfix** branch. **cleartool** or Attache determines that version **B** is the common ancestor, to be used as the base file. The merged result replaces the contents of the target, **C1.**

Figure 9 Merging from a Branch



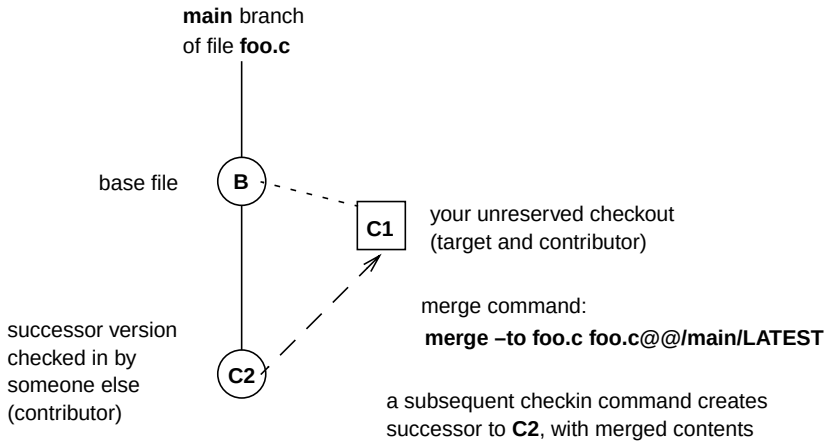
Case 2: Merging to an Unreserved Checkout

Another common use of this command arises from the *unreserved* checkout capability: you perform an unreserved checkout and edit the file, but someone else checks in a successor version ahead of you. You can check in your work only if you first merge with the version that was already checked in.

Figure 10 shows a merge in which the target is **C1**, an unreserved, checked-out version. The other contributor is **C2**, the version that was checked before **C1**. **cleartool** or Attache determines that version **B** is the common ancestor, to be used as the base file. The result of the merge replaces the

contents of the target, **C1**. **cleartool** or Attache allows **C1** to be checked in when it sees the merge arrow from **C2** to **C1.**

Figure 10 Merging into an Unreserved Checkout



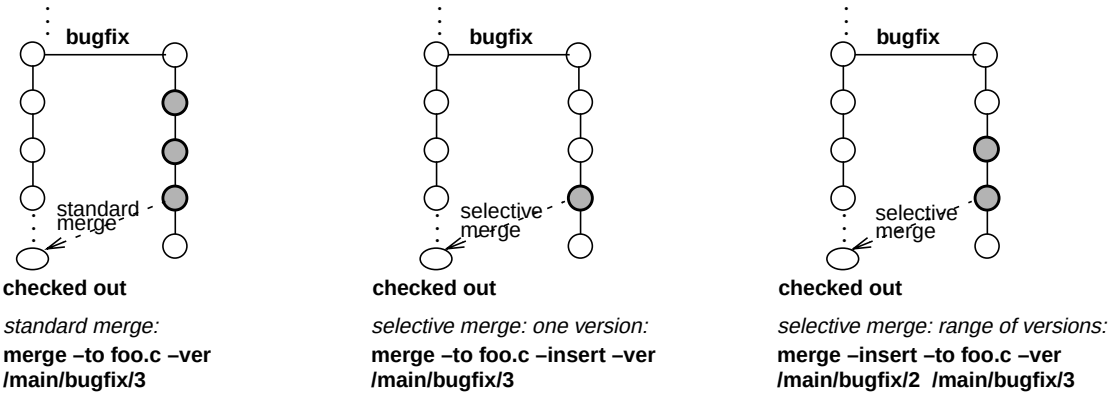
SPECIAL MERGE SCENARIOS

merge has options that invoke special kinds of merges: selective and subtractive.

Selective Merges

By default, **merge** takes into account an entire, cumulative sequence of changes. For example, a merge from version **/main/bugfix/4** to the **main** branch involves the changes made in version 3, and also the changes made in versions 2 and 1 on that branch. In some cases, however, you may want to incorporate only the changes made in one specific version (or a range of versions), disregarding the changes made in its predecessors. The **–insert** option implements a selective merge capability, as illustrated in Figure 11. In each merge, the shaded versions are the ones whose changes are merged to the **main** branch. (The commands in the following examples are wrapped to conserve space. Enter commands on a single line.)

Figure 11 Selective Merge



In a selective merge, no merge arrow is created; merge arrows indicate that all of a version's data has been merged.

Subtractive Merges

The **–delete** option invokes a subtractive merge, which is the opposite of a selective merge:

- A selective merge adds to the checked-out version the changes made in one or more other versions.
- A subtractive merge removes from the checked-out version the changes made in one or more of its predecessors.

For example, to undo the changes made in versions 5 – 9 of file **foo.c**, while retaining all the changes made before version 5 and after version 9, you can issue this command:

cmd-context **merge –to foo.c –delete –ver /main/5 /main/9** *(ClearCase and ClearCase LT only)*

cmd-context **merge –abort –to foo.c –delete –ver /main/5 /main/9** *(Attache only)*

PERMISSIONS AND LOCKS

Permissions Checking: Special permissions apply for creation of a merge arrow only. For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type.

OPTIONS AND ARGUMENTS

DESTINATION OF MERGE OUTPUT. *Default:* None.

–out *output-pname*

(File merge only) Specifies a view-private or workspace file or non-MVFS file to be the merge target. *output-pname* is not used as a contributor, and no merge arrows are created. Use this option to perform a merge that does not overwrite any of its contributors. An error occurs if *output-pname* already exists.

(Attache only) Note that *output-pname* is not uploaded to the vie. If it corresponds to a checked-out version, it remains in the workspace until it is checked in.

–to *contrib-&-result-pname*

Specifies a version of a file or directory element to be the merge target: one of the contributors to the merge, and also the location where the merged output is stored. **merge** proceeds as follows:

1. (ClearCase and ClearCase LT file merge only) Preserves the target's current contents in view-private file *contrib-&-result-pname***.contrib**. The file name may get a **.n** extension, to prevent a name collision.
2. Stores the merged output in the workspace in *contrib-&-result-pname*.

   You can suppress these data-manipulation steps by using **–ndata**; you must do so to avoid an error if the file is not checked out:

   ```
   cleartool: Error: ...
   Only a checked out version can be modified to have the data
   resulting from the merge.
   ```
3. Creates a merge arrow (hyperlink of type **Merge**) from all other contributors to the checked-out version. You can suppress this step by using the **–narrows** option.

   In ClearCase and ClearCase LT, if the merge target cannot be overwritten, **merge** saves its work in the view-private file *contrib-&-result-pname***.merge** The file name may have a **.***n* extension, to prevent a name collision.

   In Attache, if the merge target cannot be overwritten, **merge** saves its work in the workspace file *contrib-&-result-pname***.mrg**, or if that extension exists, `.m00`, `.m01`, and so on.

PERFORMING A GRAPHICAL MERGE. *Default:* Performs the merge in the command window and uses the default display font.

–g·**raphical** [**–tin**·**y**]

Performs the merge graphically. With **–tiny**, a smaller font is used to increase the amount of text displayed in each display pane.

NOTE: When merging files of type **html**, if the machine on which you execute **merge –graphical** is not the machine on which you run your HTML browser, your browser may not be able to find the pathname to the files being merged.

(CLEARCASE ONLY) USING A SEPARATE WINDOW. *Default:* Sends output to the current window.

–tin·y
Same as **–window**, but uses a smaller font in a 165-character window.

–win·dow
Displays output in a separate window, formatted as with –**columns 120**. Type an operating system interrupt character (typically, CTRL+C in the window to close it. The **merge** command returns immediately, not waiting for the window to be closed.

INTERACTIVE MERGES NOT SUPPORTED IN ATTACHE. You must specify **–ndata** or **–abort** from below.

OUTPUT FORMAT. *Default:* Displays output in the format described in the **diff** reference page.

–ser·ial_format
Reports differences with each line containing output from one contributor, instead of in a side-by-side format.

–dif·f_format
Displays output in the same style as the UNIX **diff(1)** utility.

–col·umns *n*
Establishes the overall width of side-by-side output. The default width is 80; only the first 40 or so characters of corresponding difference lines appear. If *n* does not exceed the default width, this option is ignored.

SPECIFYING THE BASE CONTRIBUTOR. *Default:* Uses the procedure described in *Determination of the Base Contributor* on page 545.

–bas·e *pname*
Specifies *pname* as the base contributor for the merge. You cannot use the **–version** option to specify this argument; use a version-extended pathname.

SPECIFYING SPECIAL MERGES. *Default:* A standard merge is performed: all the differences between the base contributor and each non-base contributor are taken into account.

–ins·ert
Invokes a selective merge of the changes made in one or more versions. See *Selective Merges* on page 549 for a description. If you specify one contributor with **–version** or a *pname* argument, only that version's changes are merged. Specifying two contributors defines an inclusive range of versions; only the changes made in that range of versions are merged.

No merge arrow is created in a selective merge.

RESTRICTIONS: You must specify the target version with the **–to** option. No version specified with **–version** or a *pname* argument can be a predecessor of the target version.

–del·ete

Invokes a subtractive merge of the changes made in one or more versions. See *Subtractive Merges* on page 550 for a description. If you specify one contributor with **–version** or a *pname* argument, only that version's changes are removed. Specifying two contributors defines an inclusive range of versions; only the changes made in that range of versions are removed.

No merge arrow is created in a subtractive merge.

RESTRICTIONS: You must specify the target version with the **–to** option. All versions specified with **–version** or a *pname* argument must be predecessors of the target version.

SUPPRESSING PARTS OF THE MERGE PROCESS. *Default:* **merge** stores its results in the workspace location specified by **–to** or **–out**; with **–to**, it also creates merge arrows.

–nda·ta

(Use only with **–to**) Suppresses the merge, but creates the corresponding merge arrows. An error occurs if you use **–ndata** along with **–out**; together, the two options leave **merge** with no work to do.

–nar·rows

(For use with **–to**; invoked by **–out**) Performs the merge, but suppresses the creation of merge arrows.

REPLACING A PREVIOUS MERGE. *Default:* An error occurs if a merge arrow is already attached to any version where **merge** would create one.

–rep·lace

Allows creation of new merge arrows to replace existing ones.

CONTROLLING USER INTERACTION. *Default:* Works as automatically as possible, prompting you to make a choice only when two or more non-base contributors differ from the base contributor.

NOTE: In Attache, the –**query** and –**qall** options are available only when performing a graphical merge (–**graphical**).

–q·uery

Turns off automatic merging for nontrivial merges and prompts you to proceed with every change in the from-versions. Changes in the to-version are automatically accepted unless a conflict exists. When you specify the **–out** option, **cleartool** uses the last pathname on the command line as the to-version.

–abo·rt

Cancels the command instead of engaging in a user interaction; a merge takes place only if it is completely automatic. If two or more nonbase contributors differ from the base contributor, a warning is issued and the command is canceled. This command is useful

in shell scripts that batch many merges (for example, all file elements in a directory) into a single procedure.

–qal·l
Turns off automated merging. **merge** prompts you to make a choice every time a nonbase contributor differs from the base contributor. This option is turned on automatically if **merge** cannot determine a common ancestor (or other base contributor), and you do not use **–base**.

SPECIFYING A COMMENT FOR THE MERGE ARROW. *Default:* Attaches a comment to each merge arrow (hyperlink of type **Merge**) with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

PASSING THROUGH OPTIONS TO THE 'MERGE' METHOD. *Default:* Does not pass any special options to the underlying **merge** method (in ClearCase and ClearCase LT, implemented by the **cleardiff** utility for all predefined element types).

–opt·ions *pass-through-options*
Allows you to specify **merge** options that are not directly supported on the **merge** command line.

If you are specifying more than one pass-through option, enclose them in quotes; **merge** must see them as a single command-line argument.

For descriptions of the options valid for ClearCase and ClearCase LT, see the **cleardiff** reference page.

For example, this **cleartool** command passes through the **–quiet** and **–blank_ignore** options:

cmd-context **merge –options "–qui –b" –to util.c /main/bugfix/LATEST/main/3**

Attache accepts the following pass-through options:

–hea·ders_only
–qui·et (mutually exclusive)
–headers_only lists only the header line of each difference. The difference lines themselves are omitted.

–quiet suppresses the file summary from the beginning of the report.

–b·lank_ignore

Ignores extra white space characters in text lines: leading and trailing white space is ignored; internal runs of white space are treated like a single SPACE character.

–vst·ack
–hst·ack

–vst·ack stacks the difference panes vertically, with the base contributor at the top.
–hst·ack displays the difference panes horizontally, with the base contributor on the left (the default behavior).

For example, this Attache command passes through the **–quiet** and **–blank_ignore** options:

cmd-context **merge –ndata –options "–qui –b" –to util.c /main/bugfix/LATEST /main/3**

SPECIFYING THE DATA TO BE MERGED. *Default:* None.

–ver·sion *contrib-version-selector ...*
(For use only if all contributors are versions of the same element) If you use the **–to** option to specify one contributor, you can specify the others with **–ver** followed by one or more version selectors. (See the **version_selector** reference page.)

contrib-pname ...
One or more pathnames, indicating the objects to be merged: versions of file elements, versions of directory elements, or any other files. If you don't use **–to**, you must specify at least two *contrib-pname* arguments.

These two commands are equivalent:

(ClearCase and ClearCase LT only)

cmd-context **merge –to foo.c –version /main/bugfix/LATEST /main/3**

cmd-context **merge –to foo.c foo.c@@/main/bugfix/LATEST foo.c@@/main/3**

(Attache only)

cmd-context **merge –nda –to foo.c –version /main/bugfix/LATEST /main/3**

cmd-context **merge –nda –to foo.c foo.c@@/main/bugfix/LATEST foo.c@@/main/3**

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Merge the version of file **util.c** in the current view or workspace with the most recent versions on the **rel2_bugfix** and **test** branches; suppress the creation of merge arrows.

  cmd-context **merge –to util.c –narrows **
  –version /main/rel2_bugfix/LATEST /main/test/LATEST *(ClearCase and ClearCase LT only)*

  cmd-context **merge –to util.c –abort –narrows –version /main/rel2_bugfix/LATEST /main/test/LATEST** *(Attache only/this command must be entered on a single line)*

- Merge the version of file **util.c**, in view **jk_fix**, to version 3 on the **main** branch, placing the merged output in a temporary file.

  cmd-context **merge –out /tmp/proj.out util.c@@/main/3 /view/jk_fix/usr/hw/src/util.c**

- Merge the version of file **util.c** to version 3 on the **main** branch, placing the merged output in a temporary file.

  cmd-context **merge –abort –out /tmp/proj.out util.c@@/main/3 util.c**

- Subtractive merge: remove the changes made in version 3 from file **util.c**.

  cmd-context **merge –to util.c –abort –delete –version util.c@@/main/3**

- (Attache only) Use **merge –graphical** to merge the file **util.c** in the current workspace with the most recent version on the **rel2_bugfix** branch.

  cmd-context **merge –to util.c –graphical –version \main\rel2_bugfix\LATEST**

SEE ALSO

describe, **diff**, **find**, **findmerge**, **rmmerge**, **update**, **xclearcase**, **xcleardiff**

mkactivity

Creates a UCM activity

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

mkact·ivity [–c·omment *c̈mment* | **–cfi·le** *pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–hea·dline** *headline*] [**–in** *stream-selector*] [**–nset**] [**–force**] [*activity-selector ...*]

DESCRIPTION

The **mkactivity** command creates a UCM activity. Activities track the work you do in completing a development task. An activity consists of a headline, which describes the task, and a change set, which identifies all versions of elements that are created or modified by work on the activity.

Each stream can have one current activity, which records any changes being made. Use **–nset** if you do not want to use an activity immediately. To begin recording changes in an activity, issue a **setactivity** command from a view that is attached to the activity's stream.

Behavior for ClearQuest-enabled Projects

When executed in a view that is associated with a ClearQuest-enabled project, this command generates an error. The correct way to create an activity is to use the **setactivity** command, specifying a ClearQuest record-ID as the *activity-selector*.

PERMISSIONS AND LOCKS

Permissions Checking: None.

Locks: An error occurs if there are locks on the following objects: the activity's UCM project VOB.

Mastership: There are no mastership requirements.

OPTIONS AND ARGUMENTS

ASSIGNING A HEADLINE TO AN ACTIVITY. *Default:* The activity's name as specified by the *activity-selector* argument.

–hea·dline *headline*
Specifies a description of the activity. The *headline* argument can be a character string of any length. Enclose a headline with special characters in double quotes. The headline is applied to all activities created with this invocation of the command.

SPECIFYING THE STREAM. *Default:* The stream attached to the current view.

–in *stream-selector*
Specifies that the activity be created in this stream.

stream-selector is of the form: **[stream:]***stream-name*[*@vob-selector*] and *vob* is the stream's UCM project VOB.

SETTING THE CURRENT ACTIVITY. *Default:* If one activity is created with this command: the newly created activity. If more than one activity is created or any number of activities is created outside a view context: none.

–nset
Specifies that the new activity not be set as the current activity for the view.

CONFIRMATION STEP. *Default:* Prompts for confirmation of a generated name for the activity if no name is specified by *activity-selector*.

–force
Suppresses the confirmation step.

NAMING THE ACTIVITY. *Default:* If one activity is created with this command: a generated name. If more than one activity is created: none.

activity-selector ...
Specifies one or more activities to create. The specifier must be unique within the project VOB.

You can specify an activity as a simple name or as an object selector of the form **[activity]:***name***@***vob-selector*, where *vob-selector* specifies a project VOB (see the **cleartool** reference page). If you specify a simple name and the current directory is not a project VOB, then this command assumes the activity resides in the project VOB associated with the stream attached to the current view. If the current directory is a project VOB, then that project VOB is the context for identifying the activity.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create an activity, but do not set it to be the current activity for the view.

  cmd-context **mkact –nset**

```
Create activity with automatically generated name?  [yes] yes
Created activity "activity990917.133218".
```

- Create an activity. The activity is created in the stream attached to the current view. Its name is generated automatically.

  cmd-context **mkact new_activity**

  ```
  Created activity "new_activity".
  Set activity "new_activity" in view "java_int".
  ```

- Create an activity whose name is generated automatically. You are not prompted for confirmation.

  cmd-context **mkact -f**

  ```
  Created activity "activity990917.134751".
  Set activity "activity990917.134751" in view "java_int".
  ```

- Create an activity with the headline "Create directories".

  cmd-context **mkactivity -headline "Create directories" create_directories**

  ```
  Created activity "create_directories".
  Set activity "create_directories" in view "webo_integ".
  ```

SEE ALSO

chactivity, **lsactivity**, **rmactivity**, **setactivity**

mkattr

Attaches attributes to objects

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- Attach attributes to specified file-system objects:

  mkattr [**–rep**·**lace**] [**–r**·**ecurse**] [**–ver**·**sion** *version-selector*]
  [**–pna**·**me**] [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery**
  | **–cqe**·**ach** | **–nc**·**omment**]
  { *attribute-type-selector value* | **–def**·**ault** *attribute-type-selector* }
  pname ...

- Attach attributes to specified non-file-system objects:

  mkattr [**–rep**·**lace**] [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery**
  | **–cqe**·**ach** | **–nc**·**omment**]
  { *attribute-type-selector value* | **–def**·**ault** *attribute-type-selector* }
  object-selector ...

- Attach attributes to versions listed in configuration record:

  mkattr [**–rep**·**lace**] [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery**
  | **–cqe**·**ach** | **–nc**·**omment**]
  [**–sel**·**ect** *do-leaf-pattern*] [**–ci**] [**–typ**·**e** { **f** | **d** } ...]
  [–**nam**·**e** *tail-pattern*] **–con**·**fig** *do-pname*
  { *attribute-type-selector value* | **–def**·**ault** *attribute-type-selector* }

DESCRIPTION

The **mkattr** command attaches an attribute to one or more objects. You can specify the objects themselves on the command line, or you can specify a particular derived object. In the latter case, **mkattr** attaches attributes to versions only—some or all the versions that were used to build that derived object.

An attribute is a name/value pair:

```
BugNum / 455              (integer-valued attribute)
BenchMark / 12.9          (real-valued attribute)
ProjectID / "orange"      (string-valued attribute)
DueOn / 5-Jan             (date-value attribute)
```

Restrictions on Attribute Use

In several situations, attempting to attach a new attribute causes a collision with an existing attribute:

- You want to change the value of an existing attribute on an object.
- (If the attribute type was created with **mkattype –vpbranch**) An attribute is attached to a version, and you want to attach an attribute of the same type to another version on the same branch.
- (If the attribute type was created with **mkattype –vpelement**) An attribute is attached to a version, and you want to attach an attribute of the same type to any other version of the element.

A collision causes **mkattr** to fail and report an error, unless you use the **–replace** option, which first removes the existing attribute.

Referencing Objects by Their Attributes

The **find** command can locate objects by their attributes. Examples:

- List all elements in the current working directory for which some version has been assigned a **BugNum** attribute.

  cmd-context **find . –element 'attype_sub(BugNum)' –print**

- List the version of element **util.c** to which the attribute **BugNum** has been assigned with the value `4059`.

  cmd-context **find util.c –version 'BugNum==4059 ' –print**

More generally, queries written in the query language can access objects using attribute types and attribute values. See the **query_language** reference page for details.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, object group member, object owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, attribute type, object to which the attribute is being attached (for non-file-system objects).

OPTIONS AND ARGUMENTS

MOVING AN ATTRIBUTE OR CHANGING ITS VALUE. *Default:* An error occurs if an attribute collision occurs (see *Restrictions on Attribute Use* on page 561).

–rep·lace
Removes an existing attribute of the same type before attaching the new one, thus avoiding the collision. (No error occurs if a collision would not have occurred.)

SPECIFYING THE ATTRIBUTE TYPE AND VALUE. *Default:* None. You must specify an existing attribute type; you must also indicate a value, either directly or with the **–default** option.

attribute-type-selector
An attribute type, previously created with **mkattype**. The attribute type must exist in each VOB containing objects to which you are applying attributes, or (if *attribute-type-selector* is a global type) in the Admin VOB hierarchy associated with each VOB. Specify *attribute-type-selector* in the form [**attype:**]*type-name*[**@***vob-selector*]

type-name	Name of the attribute type
vob-selector	Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any files-ystem object within the VOB (if the VOB is mounted)

–def·ault
If the attribute type was created with a default value (**mkattype –default**), you can use **–default** *attribute-type-name* to specify the name/value pair. An error occurs if the attribute type was not created with a default value.

value
Specifies the attribute's value. The definition of the attribute type specifies the required form of this argument (for example, to an integer). It may also restrict the permissible values (for example, to values in the range 0–7).

Value Type	Input Format
integer	Any integer that can be parsed by the **strtol(2)** system call
real	Any real number that can be parsed by the **strtod(2)** system call

date A date-time string in one of the following formats:
date.*time* | *date* | *time* | **now**
where

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** | **yesterday** | **Sunday** |...| **Saturday** | **Sun** |...| **Sat**
long-date	:=	*d*[*d*]**-***month*[**-**[*yy*]*yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit *date*, the default is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want to resolve the time to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets,Greenwich Mean Time (GMT) is used. (Dates before January 1, 1970 UTC are invalid.)

string Any string in standard C-language string literal format. It must be enclosed in double quotes; it can include escape sequences: **\n**, **\t**, and so on.

NOTE: The double-quote (**"**) character is special to both the **cleartool** command processor and the UNIX shells. Thus, you must escape or quote this character on the command line. These two commands are equivalent:
cleartool mkattr QAed '"TRUE"' hello.c
cleartool mkattr QAed \"TRUE\" hello.c

opaque A word consisting of an even number of hexadecimal digits (for example, `04a58f` or `FFFB`). The value is stored as a byte sequence in a host-specific format.

DIRECTLY SPECIFYING THE OBJECTS. The options and arguments in this section specify objects to be assigned attributes directly on the command line. Do not use these options and arguments when using a derived object to provide a list of versions to be assigned attributes.

object-selector ...
(Required) One or more names of objects to be assigned attributes. Specify *object-selector* in one of the following forms:

vob-selector	**vob:***pname-in-vob*

pname-in-vob can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted). It cannot be the pathname of the *VOB storage directory*.

attribute-type-selector	**attype:***type-name*[**@***vob-selector*]
branch-type-selector	**brtype:***type-name*[**@***vob-selector*]
element-type-selector	**eltype:***type-name*[**@***vob-selector*]
hyperlink-type-selector	**hltype:***type-name*[**@***vob-selector*]
label-type-selector	**lbtype:***type-name*[**@***vob-selector*]
trigger-type-selector	**trtype:***type-name*[**@***vob-selector*]
pool-selector	**pool:***pool-name*[**@***vob-selector*]
hlink-selector	**hlink:***hlink-id*[**@***vob-selector*]
oid-obj-selector	**oid:***object-oid*[**@***vob-selector*]

The following object selector is valid only if you use MultiSite:

replica-selector	**replica:***replica-name*[**@***vob-selector*]

[**–pna·me**] *pname ...*

(Required) One or more pathnames, indicating objects to be assigned attributes. If *pname* has the form of an object selector, you must include the **–pname** option to indicate that *pname* is a pathname.

- A standard or view-extended pathname to an element specifies the version in the view.
- A version-extended pathname specifies an element, branch, or version, independent of view.

Examples:

`foo.c`	*(version of 'foo.c' selected by current view)*
`/view/gamma/usr/project/src/foo.c`	*(version of 'foo.c' selected by another view)*
`foo.c@@/main/5`	*(version 5 on main branch of 'foo.c')*
`foo.c@@/REL3`	*(version of 'foo.c' with version label 'REL3')*
`foo.c@@`	*(the element 'foo.c')*
`foo.c@@/main`	*(the main branch of element 'foo.c')*

Use **–version** to override these interpretations of *pname*.

–ver·sion *version-selector*

For each *pname*, attaches the attribute to the version specified by *version-selector*. This option overrides both version-selection by the view and version-extended naming. See the **version_selector** reference page for syntax details.

–r·ecurse
Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are not traversed during the recursive descent into the subtree.

NOTE: **mkattr** differs from some other commands in its default handling of directory element *pname* arguments: it assigns an attribute to the directory element itself; it does not assign attributes to the elements cataloged in the directory.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

USING A DERIVED OBJECT TO SPECIFY VERSIONS. The options and arguments in this section specify versions to be assigned attributes by selecting them from the configuration records associated with a particular derived object. Do not use these options when specifying objects to be assigned attributes directly on the command line.

–con·fig *do-pname*
(Required) Specifies one derived object. A standard pathname or view-extended pathname specifies the DO that currently appears in a view. To specify a DO independent of view, use an extended name that includes a *DO-ID* (for example, **hello.o@@24–Mar.11:32.412)** or a version-extended pathname to a DO version.

With the exception of checked-out versions, **mkattr** attaches attributes to all the versions that would be included in a **catcr –flat** listing of that derived object. Note that this includes any DO created by the build and subsequently checked in as a DO version.

If the DO's configuration includes multiple versions of the same element, the attribute is attached only to the most recent version.

Use the following options to modify the list of versions to which attributes are attached.

–sel·ect *do-leaf-pattern*
–ci
–nam·e *tail-pattern*
–typ·e { **f** | **d** } ...
Modify the set of versions to be assigned attributes in the same way that these options modify a **catcr** listing. For details, see the **catcr** reference page and the *EXAMPLES* section.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create an attribute type named **BugNum**. Then, attach that attribute with the value 21 to the version of **util.c** that fixes bug 21.

  cmd-context **mkattype –nc –vtype integer BugNum**
  ```
  Created attribute type "BugNum".
  ```

  cmd-context **mkattr BugNum 21 util.c**
  ```
  Created attribute "BugNum" on "util.c@@/main/maintenance/3".
  ```

- Attach a **TESTED** attribute to the version of **hello.h** in the view, assigning it the value `"TRUE"`.

  cmd-context **mkattr TESTED "'TRUE"' hello.h**
  ```
  Created attribute "TESTED" on "hello.h@@/main/2".
  ```

- Update the value of the **TESTED** attribute on **hello.h** to "`FALSE`". This example shows that to overwrite an existing attribute value, you must use the **–replace** option.

  cmd-context **mkattr –replace TESTED ’"FALSE"’ hello.h**
  ```
  Created attribute "TESTED" on "hello.h@@/main/2".
  ```

- Attach a **RESPONSIBLE** attribute to the *element* (not a particular version) **hello.c**.

  cmd-context **mkattr RESPONSIBLE ’"Anne"’ hello.c@@**
  ```
  Created attribute "RESPONSIBLE" on "hello.c@@".
  ```

- Attach a **TESTED_BY** attribute to the version of **util.c** in the view, assigning it the value of the **USER** environment variable as a double-quoted string. Using **\"** causes the shell to pass through (to **cleartool**) the double-quote character instead of interpreting it. (Specifying the attribute value as `'"$USER"'` does not work, because the single quotes suppress environment variable substitution.)

  cmd-context **mkattr TESTED_BY \"$USER\" util.c**
  ```
  Created attribute "TESTED_BY" on "util.c@@/main/5".
  ```

- Attach a **TESTED** attribute to the version of **foo.c** in the current view, specifying an attribute string value that includes a space.

  cmd-context **mkattr TESTED ’"NOT TRUE"’ foo.c**
  ```
  Created attribute "TESTED" on "foo.c@@/main/CHECKEDOUT".
  ```

- Attach a **TESTED** attribute with the default value to each version that was used to build derived object **hello.o**. Note that the attribute is assigned to versions of both files and directories.

  cmd-context **mkattr –config hello.o -default TESTED**
  ```
  Created attribute "TESTED" on "/usr/hw/@@/main/1".
  Created attribute "TESTED" on "/usr/hw/src@@/main/2".
  Created attribute "TESTED" on "/usr/hw/src/hello.c@@/main/3".
  Created attribute "TESTED" on "/usr/hw/src/hello.h@@/main/1".
  ```

- Attach a **TESTED** attribute with the value `"FALSE"` to those versions that were used to build **hello**, and whose pathnames match the ***.c** tail pattern.

  cmd-context **mkattr –config 'hello' –name '*.c' TESTED "'FALSE"'**
  ```
  Created attribute "TESTED" on "/usr/hw/src/hello.c@@/main/3".
  Created attribute "TESTED" on "/usr/hw/src/util.c@@/main/1".
  ```

- Attach a **TESTED** attribute with the value `"TRUE"` to all versions in the VOB mounted at **/src/lib** that were used to build **hello**. Use interactive mode to enable use of the "**...**" wildcard.

  `%` **cleartool**

  `cleartool>` **mkattr –config hello –name '/src/lib/...' TESTED "'TRUE"'**
  ```
  Created attribute "TESTED" on "/src/lib/hello.c@@/main/8".
  Created attribute "TESTED" on "/src/lib/util.c@@/main/5".
  Created attribute "TESTED" on "/src/lib/hello.h@@/main/1".
  ```

SEE ALSO

describe, **mkattype**, **query_language**, **rmattr**

mkattype

Creates or updates an attribute type object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkattype [**–rep**·**lace**] [**–glo**·**bal** [**–acq**·**uire**] | **–ord**·**inary**]
[**–vpe**·**lement** | **–vpb**·**ranch** | **–vpv**·**ersion**] [**–sha**·**red**]
[**–vty**·**pe** { **integer** | **real** | **time** | **string** | **opaque** }]
[[**–gt** *low-val* | **–ge** *low-val*] [**–lt** *high-val* | **–le** *high-val*]
| **–enu**·**m** *value*[,...]]
[**–def**·**ault** *default-val*]
[**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
attribute-type-selector ...

DESCRIPTION

The **mkattype** command creates one or more attribute types for future use within a VOB. After creating an attribute type in a VOB, you can use **mkattr** to attach attributes of that type to objects in that VOB.

Attributes as Name/Value Pairs

An attribute is a name/value pair. When creating an attribute type, you must specify the kind of value (integer, string, and so on). You can also restrict the possible values to a particular list or range. For example:

- Attributes of type **FUNC_TYPE** could be restricted to integer values in the range 1–5
- Attributes of type **QAed** colud be restricted to the string values **TRUE** and **FALSE**.

Predefined Attribute Types

Each new VOB is created with two string-valued attributes types, named **HlinkFromText** and **HlinkToText**. When you enter a **mkhlink –ftext** command, the from-text you specify is stored as an instance of **HlinkFromText** on the hyperlink object. Similarly, an **HlinkToText** attribute implements the to-text of a hyperlink.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following (for **–replace** only): type owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, attribute type (for **–replace** only).

OPTIONS AND ARGUMENTS

HANDLING OF NAME COLLISIONS. *Default:* An error occurs if an attribute type named *type-name* already exists in the VOB.

–rep·lace

Replaces the existing definition of *type-name* with a new one. If you do not include options from the existing definition, their values are replaced with the defaults (Exception: the type's scope does not change; you must explicitly specify **–global** or **–ordinary**).

If you specify a comment when using **–replace**, the comment appears in the event record for the modification (displayed with **lshistory –minor**); it does not replace the object's creation comment (displayed with **describe**). To change an object's creation comment, use **chevent**.

Constraints:

- If there are existing attributes of this type, you cannot change the **–vtype** value.
- If there are existing attributes of this type or if the containing VOB is replicated, you cannot replace a less restrictive **–vpelement**, **–vpbranch**, or **–vpversion** specification with a more restrictive one. (**–vpelement** is the most restrictive.)
- You cannot replace the predefined attribute types **HlinkFromText** and **HlinkToText**.
- When replacing an attribute type that was created with the **–shared** option, you must use **–shared** again; that is, you cannot convert an attribute type from shared to unshared.
- When converting a global type to ordinary, you must specify the global type as the *attribute-type-selector* argument. You cannot specify a local copy of the global type.

SPECIFYING THE SCOPE OF THE ATTRIBUTE TYPE. *Default:* Creates an ordinary attribute type that can be used only in the current VOB.

–glo·bal [**–acq·uire**]

Creates an attribute type that can be used as a global resource by client VOBs in the administrative VOB hierarchy. With **–acquire**, **mkattype** checks all eclipsing types in client VOBs and converts them to local copies of the new global type.

For more information, see *Administering ClearCase*.

–ord·inary
Creates an attribute type that can be used only in the current VOB.

INSTANCE CONSTRAINTS. *Default:* In a given element, one attribute of the new type can be attached to each version, to each branch, and to the element itself. One attribute of the type can be attached to other types of VOB objects.

–vpe·lement
Attributes of this type can be attached only to versions; and only one version of a given element can get an attribute of this type.

–vpb·ranch
Attributes of this type can be attached only to versions; and only one version on each branch of a given element can get an attribute of this type.

–vpv·ersion
Attributes of this type can be attached only to versions; within a given element, all versions can get an attribute of this type.

SPECIFYING THE KIND OF VALUE. *Default:* One or more string-valued attribute types are created.

–vty·pe integer
Attributes of this type can be assigned integer values. You can use these options to restrict the possible values: **–gt**, **–ge**, **–lt**, **–le**, **–enum**.

–vty·pe real
Attributes of this type can be assigned floating-point values. You can use these options to restrict the possible values: **–gt**, **–ge**, **–lt**, **–le**, **–enum**.

–vty·pe time
Attributes of this type can be assigned values in the date-time format described in the **mkattr** reference page. You can use these options to restrict the possible values: **–gt**, **–ge**, **–lt**, **–le**, **–enum**.

–vty·pe string
Attributes of this type can be assigned character-string values. You can use the **–enum** option to restrict the possible values.

–vty·pe opaque
Attributes of this type can be assigned arbitrary byte sequences as values.

MASTERSHIP OF THE ATTRIBUTE TYPE. *Default:* Attempts to attach or remove attributes of this type succeeda only in the *VOB replica* that is the current *master* of the attribute type. The VOB replica in which the new attribute type is created becomes its initial master.

–sha·red
If you specify –**vpbranch**, –**vpelement**, or –**vpversion**, ClearCase and ClearCase LT check the mastership of the branch, element, or version's branch to which you attach or remove the attribute when you invoke the **mkattr** or **rmattr** command. If you do not specify –**vpbranch**, –**vpelement**, or –**vpversion**, and the object to which you attach or remove the attribute is a version, mastership of the branch is checked when you invoke the **mkattr** or **rmattr** command. If you do not specify –**vpbranch**, –**vpelement**, or –**vpversion**, and the object to which you attach or remove the attribute is not a version, the mastership of the object is checked when you invoke the **mkattr** or **rmattr** command.

RESTRICTING THE POSSIBLE VALUES. *Default:* The values that can be assigned to attributes of the new type are unrestricted within the basic value type (any integer, any string, and so on). You can specify a list of permitted values, using **–enum**; alternatively, you can specify a range using **–gt** or **–ge** to specify the lower bound, and **–lt** or **–le** to specify the upper bound.

–gt *low-val* or **–ge** *low-val*
Lower bound of an integer, real, or time value. **–gt** means greater than. **–ge** means greater than or equal to.

–lt *high-val* or **–le** *high-val*
Upper bound of an integer, real, or time value. **–lt** means less than. **–le** means less than or equal to.

–enu·m *value*[,...]
Comma-separated list (no white space allowed) of permitted values for any value type. See the description of the *value* argument in the **mkattr** reference page for details on how to enter the various kinds of *value* arguments.

SPECIFYING A DEFAULT ATTRIBUTE VALUE. *Default:* You cannot use **mkattr –default** to create an instance of this attribute type; you must specify an attribute value on the command line.

–def·ault *default-val*
Specifies a default attribute value; entering a **mkattr –default** command creates an attribute with the value *default-val*.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

NAMING THE ATTRIBUTE TYPES. *Default:* The attribute type is created in the VOB that contains the current working directory unless you specify another VOB with the *@vob-selector* argument.

attribute-type-selector ...

Names of the attribute type(s) to be created. Specify *attribute-type-selector* in the form [**attype:**]*type-name*[**@***vob-selector*]

type-name — Name of the attribute type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a string-valued attribute type named **Responsible**.

  cmd-context **mkattype -nc Responsible**
  ```
  Created attribute type "Responsible".
  ```

- Create an integer-valued attribute type named **Confidence_Level**, with a low value of 1 and a high value of 10. Constrain its use to one per branch.

  cmd-context **mkattype -nc -vpbranch -vtype integer -gt 0 -le 10 Confidence_Level**
  ```
  Created attribute type "Confidence_Level".
  ```

- Create a string-valued attribute type named **QAed**, with an enumerated list of valid values.

  cmd-context **mkattype -nc -enum "'TRUE','FALSE','in progress'" QAed**
  ```
  Created attribute type "QAed".
  ```

- Create a time-valued attribute type named **QA_date**, with the current date as the default value. Provide a comment on the command line.

  cmd-context **mkattype -c "attribute for QA date" -vtype time -default today QA_date**
  ```
  Created attribute type "QA_date".
  ```

- Create an enumerated attribute type, with a default value, called **Released**.

  cmd-context **mkattype -nc -enum "'TRUE','FALSE'" -default "'FALSE'" Released**
  ```
  Created attribute type "Released".
  ```

- Change the default value of an existing attribute type named **TESTED**. Provide a comment on the command line.

  cmd-context **mkattype -replace -default "'TRUE'" -c "changing default value" TESTED**
  ```
  Replaced definition of attribute type "TESTED".
  ```

SEE ALSO

lstype, **mkattr**, **rename**, **rmattr**, **type_object**

mkbl

Creates a UCM baseline or set of baselines

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

- Create a baseline of a component or set of baselines of components:
  mkbl [–c·omment *comment* | **–cfi· le** *pname* | **–cq· uery** | **–nc·omment]**
  [–vie·w *view-tag* **]**
  [–com·ponent *component-selector*[,...] | **–all** | –**act·ivities** *activity-selector*[,...] **]**
  [–ide·ntical]
  [–nlabel | **–inc·remental** | **–fu·ll]**
  baseline-root-name
- Create a baseline by importing a label type:
  mkbl [–c·omment *comment* | **–cfi· le** *pname* | **–cq· uery** | **–nc·omment]**
  –imp·ort *label-type-selector ...*

DESCRIPTION

The **mkbl** command creates a *baseline* or set of baselines. A baseline represents a snapshot of the changes made to a particular component in the context of a particular stream—it is a version of a component. For each element in the component, the baseline records the version of that element selected by the stream's configuration at the time the **mkbl** operation is executed. The baseline also records the list of activities in the stream whose changes sets contain versions of the component's elements.

A baseline selects one version of each element of a component. You can create multiple baselines per component, just as you can create multiple versions of an element. A baseline is associated with only one component and you can only create one baseline per component per invocation of **mkbl**.

By default, all components that have been modified since the last full baseline are considered as candidates for new baselines. You can also create baselines for a subset of components in the stream or for components modified by specific activities.

Initial Baseline

When you create a component, it includes an initial baseline whose name is of the form *component-name*_**INITIAL**. This baseline selects the **/main/0** version of the component's root directory and serves as a starting point for successive baselines of the component.

Creating a Baseline for an Unmodified Component

Use the –**identical** option to create a new baseline for a component that has not been modified. This can be useful in working with several components. You can create new baselines for a set of components independent of whether they have been modified.

Creating Baselines that Include a Set of Activities

By default, all activities modified since the last baseline was made are captured in new baselines. You can select a subset of activities for inclusion in the baseline. If there are dependencies between the change sets of activities, you may not be able to include just the activity you want; you'll also need to include those activities that it depends on.

A single baseline is created if the selected activities are part of the same component. If an activity modifies more than one component, a new baseline is created for each component it modifies.

Creating a Baseline by Importing a Label

You can recognize a VOB as a component with the **mkcomp** command. When you do this, the VOB is given an initial baseline that selects the **/main/0** version of the component root directory. However, this baseline does not give you access to files and directories that are already in the VOB.

You can create a new baseline that corresponds to a set of labeled versions in the VOB. To do this, use the –**import** option, specifying a label-type-selector. The **mkbl** command creates a baseline that selects the labeled versions, making them accessible to the UCM project.

Before creating the baseline, be sure that the label is unlocked and ordinary (not global) and that labeled elements are checked in. The label is locked when the baseline is created and you cannot move the label later. Be certain the label selects a version of all visible elements.

Baseline Names

Baseline identifiers are made up of two parts: a user-specifiable root name and a generated, unique numeric extension. The same root name can be used for baselines of more than one component. However, a root name can be used only once per component per stream.

When you create a baseline by importing a label, the root name is derived from the label's type selector. For example, the label-type selector **REL1@/vobs/baz** generates a baseline root name of **REL1** whose scope is the **baz** component.

Baseline Labels

You can choose whether versions of the baseline are to be labeled when the baseline is created. Baselines can be unlabeled, incrementally labeled, or fully labeled.

All baselines record a component's current configuration in a stream, but only labeled baselines can be used to configure other streams (via the **rebase** operation or **mkstream**).

Choose a labeling scheme that suits your project's structure. Incremental baselines are typically faster to create than full baselines. Specifically, the time required to create a baseline is as follows:

- For a full baseline, it is proportional to the number of elements in the component.
- For an incremental baseline, the time is proportional to the number of elements changed since the last full baseline.

These options control labeling during baseline creation:

- The **–nlabel** option, which creates an unlabeled baseline. Unlabeled baselines cannot be used as foundation baselines to configure a stream. They can be used with the **diffbl** command.
- The **–incremental** option, which labels versions of elements that have changed since the last full baseline was created.
- The **–full** option, which creates a baseline by selecting and labeling a version of each element in the component.

You can change the labeling status for a baseline with the **chbl** command.

Promotion Levels

Baselines are marked with a promotion level that signifies the quality of the baseline. When created, a project VOB is assigned an ordered set of promotion levels, one of which is designated the default promotion level, the level assigned to new baselines when they are created.

See the **setplevel** command for further information.

PERMISSIONS

Permissions Checking: No special permissions are required.

Locks: An error is generated if there are locks on any of the following objects: UCM project VOB, component, containing stream, label.

Mastership: The master replica of the indicated objects must match the replica (originally) performing the operation.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cq**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE VIEW AND STREAM. Default: The stream to which the current view is attached.

·**–vie·w** *view-tag*
Specifies the view from which to create baselines. Baselines are created in the stream that the view is attached to.

For example, if you are working in **coyne_dev_view**, but want to create a baseline from the configuration specified by the view **coyne_integration_view**, use **–view coyne_integration_view**. This creates a baseline in the project's integration stream that includes all the checked-in versions contained in **coyne_integration_view**. If you do not specify *view-tag*, the current view is used.

SPECIFYING THE COMPONENTS OR ACTIVITIES. *Default:* **–all**.

–com·ponent *component-selector*[,...]
Specifies the components for which baselines are made.

component-selector is of the form: [**component:**]*component-name*[*@vob-selector*] and *vob* is the component's UCM project VOB.

–all
Creates a baseline for each component in the project that has been modified since the last baseline.

–ide·ntical
Creates new baselines for all components, regardless of whether they have been modified.

–act·ivities *activity-selector, ...*
Specifies a list of activities to include in the new baselines.

activity-selector is of the form: [**activity:**]*activity-name*[*@vob-selector*] where *vob* is the activity's UCM project VOB.

By default, all activities with changes that are not recorded in the last baselines are recorded in the new baselines. You can use this option to include only a subset of the unrecorded changes in the new baselines. A baseline is created for each component that has unrecorded changes in the specified list of activities.

The list of activities must be complete. That is, they must not depend on the inclusion of any other activities. Activity **A2** is dependent on activity **A1** if they both contain versions of the same element and **A2** contains a later version than **A1**. If the list of activities is incomplete, the required activities are listed and the operation fails.

SELECTING LABELING BEHAVIOR. *Default:* **–incremental**.

–nla·bel
Specifies that versions for this baseline are not labeled. Unlabeled baselines cannot be used as foundation baselines, but can be used by the **diffbl** command and labeled later.

–inc·remental
Labels only versions that have changed since the last full baseline was created.

–fu·ll
Labels all versions visible below the component's root directory.

SPECIFYING THE BASELINE ROOT. *Default*: None.

baseline-root-name
Specifies the root portion of the baseline name. See *Baseline Names* on page 575.

SPECIFYING A LABEL TO IMPORT. *Default:* None.

–imp·ort *label-type-selector*
Creates a baseline using versions marked with the specified *label-type-selector.* The label type must be applied to the component's root directory and to every element below the root directory that you want to include in the component. Baselines are created as successors to the initial baseline. The scope of the label type must be ordinary, not global.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a baseline for a component **xroutines** by importing a label type.

cmd-context **mkbl -c "Import BL2 label" –import BL2@/vobs/xroutines**

- Create baselines for all components in the project that have been modified since the last baseline was created.

  cmd-context **mkbl BL1**

  ```
  Created baseline "BL1.119" in component "webo_modeler".
  Begin incrementally labeling baseline "BL1.119".
  Done incrementally labeling baseline "BL1.119".
  Created baseline "BL1.120" in component "webo_gui".
  Begin incrementally labeling baseline "BL1.120".
  Done incrementally labeling baseline "BL1.120".
  ```

- Create baselines for the components modified by a particular activity.

  cmd-context **mkbl -activities line-lib@/vobs/pvob1**

SEE ALSO

chbl, **diffbl**, **lsbl**, **rmbl**

mkbranch

Creates a new branch in the version tree of an element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkbranch [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery**
| **–cqe·ach** | **–nc·omment**] [**–nwa·rn**]
[**–nco**] [**–ver·sion** *version-selector*] *branch-type-selector pname ...*

DESCRIPTION

The **mkbranch** command creates a new branch in the version trees of one or more elements. The new branch is checked out, unless you use the **–nco** option. In Attache, after the command is executed, any files checked out successfully are downloaded to the workspace.

Auto-Make-Branch

The **checkout** command sometimes invokes **mkbranch** automatically. If the view's version of an element is selected by a config spec rule with a **–mkbranch** *branch-type* clause, **checkout** does the following:

1. Creates a branch of type *branch-type*.
2. Checks out (version 0 on) the newly created branch.

Similarly, entering a **mkbranch** command explicitly can invoke one or more *additional* branch-creation operations. See *Multiple-Level Auto-Make-Branch* in the **checkout** reference page.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, branch type, element, pool (nondirectory elements only).

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SUPPRESSING WARNING MESSAGES *Default:* Warning messages are displayed.

–nwa·rn
Suppresses warning messages.

CHECKOUT OF THE NEW BRANCH. *Default:* The newly created branch is checked out. Additional checkouts may ensue; see the *Auto-Make-Branch* section.

–nco
Suppresses automatic checkout of the branch. In Attache, this option also suppresses downloading of the files to the workspace.

SPECIFYING THE BRANCH TYPE. *Default:* None.

branch-type-selector
An existing branch type, previously created with **mkbrtype**. The branch type must exist in each VOB in which you are creating a branch, or (if *branch-type-selector* is a global type) in the Admin VOB hierarchy associated with each VOB. Specify *branch-type-selector* in the form [**brtype:**]*type-name*[**@***vob-selector*]

type-name	Name of the branch type See the *Object Names* section in the **cleartool** reference page for rules about composing names.	
vob-selector	VOB specifier	
	pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

SPECIFYING THE BRANCH POINTS. *Default:* None.

–ver·sion *version-selector*
For each *pname*, creates the branch at the version specified by *version-selector*. This option overrides both version-selection by the view and version-extended naming. See the **version_selector** reference page for syntax details.

pname ...
One or more pathnames, indicating the versions at which branches are to be created.

- A standard or view-extended pathname to an element specifies the version in the view.
- A version-extended pathname specifies a version, independent of view.

Use **–version** to override these interpretations of *pname*.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a branch type named **bugfix**. Then, set a view (in Attache, a workspace) with a config spec that prefers versions on the **bugfix** branch, and create a branch of that type in file **util.h.**

  cmd-context **mkbrtype –c "bugfixing branch" bugfix**

  ```
  Created branch type "bugfix".
  ```

  cmd-context **setview smg_bugfix** *(ClearCase and ClearCase LT only)*

  cmd-context **setws smg_bugfix** *(Attache only)*

  cmd-context **mkbranch –nc bugfix util.h**

  ```
  Created branch "bugfix" from "util.h" version "/main/1".
  Checked out "util.h" from version "/main/bugfix/0".
  ```

- Create a branch named **rel2_bugfix** off the version of **hello.c** in the view, and check out the initial version on the branch.

  cmd-context **mkbranch –nc rel2_bugfix hello.c**

  ```
  Created branch "rel2_bugfix" from "hello.c" version "/main/4".
  Checked out "hello.c" from version "/main/rel2_bugfix/0".
  ```

- Create a branch named **maintenance** off version **/main/1** of file **util.c**. Do not check out the initial version on the branch.

  cmd-context **mkbranch –version /main/1 –nco –nc maintenance util.c**

  ```
  Created branch "maintenance" from "util.c" version "/main/1".
  ```

- Create a branch named **bugfix** off version **/main/3** of file **hello.c**, and check out the initial version on the branch. Use a version-extended pathname to specify the version.

cmd-context **mkbranch –nc bugfix hello.c@@/main/3**

```
Created branch "bugfix" from "hello.c" version "/main/3".
Checked out "hello.c" from version "/main/bugfix/0".
```

- For each file with a **.c** extension, create a branch named **patch2** at the currently selected version, but do not check out the initial version on the new branch. Provide a comment on the command line.

  cmd-context **mkbranch –nco –c "release 2 code patches" patch2 *.c**

```
Created branch "patch2" from "cm_add.c" version "/main/1".
Created branch "patch2" from "cm_fill.c" version "/main/3".
Created branch "patch2" from "msg.c" version "/main/2".
Created branch "patch2" from "util.c" version "/main/1".
```

SEE ALSO

mkbrtype, **rename**

mkbrtype

Creates/updates a branch type object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkbrtype [–rep·lace] [–glo·bal [–acq·uire] | –ord·inary] [–pbr·anch]
[–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment]**
branch-type-selector ...

DESCRIPTION

The **mkbrtype** command creates one or more *branch types* with the specified names for future use within a particular VOB. After creating a branch type in a VOB, you can create branches of that type in that VOB's elements, using **mkbranch**.

Instance Constraints

The version-extended naming scheme requires that a branch of a version tree have at most one subbranch of a given type. (If there were two **bugfix** subbranches of the **main** branch, the version-extended pathname **foo.c@@/main/bugfix/3** would be ambiguous.) However, by default only one branch of this type can be created in an element's entire version tree. The **–pbranch** option loosens this constraint.

Recommended Naming Convention

A VOB cannot contain a branch type and a label type with the same name. For this reason, we strongly recommend that you adopt this convention:

- Make all letters in names of branch types lowercase (**a** – **z**).
- Make all letters in names of label types uppercase (**A** – **Z**).

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following (with **–replace** only): type owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, branch type (with **–replace** only).

OPTIONS AND ARGUMENTS

HANDLING OF NAME COLLISIONS. *Default:* An error occurs if a branch type named *type-name* already exists in the VOB.

–rep·lace
Replaces the existing definition of *type-name* with a new one. If you do not include options from the existing definition, their values are replaced with the defaults (Exception: the type's scope does not change; you must explicitly specify **–global** or **–ordinary**).

If you specify a comment when using **–replace**, the comment appears in the event record for the modification (displayed with **lshistory –minor**); it does not replace the object's creation comment (displayed with **describe**). To change an object's creation comment, use **chevent**.

Constraints:

- You cannot replace the predefined branch type **main**.
- If there are existing branches of this type or if the containing VOB is replicated, you cannot replace a less constrained definition (**–pbranch** specified) with a more constrained definition (omitting the **–pbranch** option).
- When converting a global type to ordinary, you must specify the global type as the *branch-type-selector* argument. You cannot specify a local copy of the global type.

SPECIFYING THE SCOPE OF THE BRANCH TYPE. *Default:* Creates an ordinary branch type that can be used only in the current VOB.

–glo·bal [**–acq·uire**]
Creates a branch type that can be used as a global resource by client VOBs in the administrative VOB hierarchy. With **–acquire**, **mkbrtype** checks all eclipsing types in client VOBs and converts them to local copies of the new global type.

For more information, see *Administering ClearCase*.

–ord·inary
Creates a branch type that can be used only in the current VOB.

INSTANCE CONSTRAINTS. *Default:* Only one branch of the new type can be created in a given element's version tree.

–pbr·anch
Multiple branches of the same type can be created in the version tree, but they must be created off different branches.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE BRANCH TYPES. *Default:* The branch type is created in the VOB that contains the current working directory unless you specify another VOB with the **@***vob-selector* argument.

branch-type-selector...
Names of the branch types to be created. Specify *branch-type-selector* in the form [**brtype:**]*type-name*[**@***vob-selector*]

type-name — Name of the branch type
See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — VOB specifier
Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

Also see the section *Recommended Naming Convention* on page 584.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a branch type named **bugfix_v1**, which can be used only once in an element's version tree. Provide a comment on the command line.

  cmd-context **mkbrtype –c "bugfix development branch for V1" bugfix_v1**

  ```
  Created branch type "bugfix_v1".
  ```

- Create two branch types for working on program patches, and a bugfix branch for release 2. Constrain their use to one per branch.

cmd-context **mkbrtype –nc –pbranch patch2 patch3 rel2_bugfix**

```
Created branch type "patch2".
Created branch type "patch3".
Created branch type "rel2_bugfix".
```

- Change the constraint on an existing branch type so that it can be used only once per branch. Provide a comment on the command line.

  cmd-context **mkbrtype -replace -pbranch -c "change to one per branch" bugfix_v1**

  ```
  Replaced definition of branch type "bugfix_v1".
  ```

SEE ALSO

chtype, **describe**, **lstype**, **mkbranch**, **rename**, **rmtype**, **type_object**

mkcomp

Creates a component object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

mkcomp [–**c**·**omment** *comment* | –**cfi**·**le** *pname* | –**cq**·**uery** | –**nc**·**omment** **]**
–**root** *dir-pname component-selector*

DESCRIPTION

The **mkcomp** command creates a UCM component. A component groups directories and file elements. The scope of a UCM project is declared in terms of components. A project must contain at least one component, and it can contain multiple components. Projects can share components.

This command must be used within a view context.

Component objects live in project VOBs, and point to directory elements. All elements below the directory root are in the component.

An initial baseline is automatically created when you create a component. This baseline selects the **/main/0** version of the component's root directory. Use this as a starting point for making changes to the component.

PERMISSIONS

Permissions: No special permissions are required to create a component.

Locks: An error occurs if there are locks on the following objects: UCM project VOB.

Mastership: The master replica of the indicated objects must match the replica (originally) performing the operation

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·**omment** *comment* | **–cfi**·**le** *comment-file-pname* | **–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**
Overrides the default with the option you specify. See the **comments** reference page.

The comment is stored in the creation event of the component object.

SPECIFYING A COMPONENT SELECTOR AND LOCATION.

–**root** *dir-pname*
Specifies the root directory pathname for this component. The *directory-pathname* must be the root directory of a VOB. A VOB directory can be referenced only by one component in one project VOB.

component-selector
Identifies the component.

component-selector is of the form [**component:**]*component-name*[@*vob-selector*] where *vob* is the component's UCM project VOB.

If no *vob-selector* is given, the component is created in the project VOB if it contains the current working directory, otherwise the component is not created.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a component.

cmd-context **mkcomp -c "modeling component" **
-root /vobs/webo_modeler webo_modeler@/vobs/webo_pvob

```
Set Admin VOB for component "webo_modeler"
Created component "webo_modeler".
```

SEE ALSO

lscomp, **mkbl**, **rmcomp**

mkdir

Creates a directory element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkdir [–nco] [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**] *dir-pname ...*

DESCRIPTION

NOTE: A new directory element can be created only if its parent directory is checked out. **mkdir** appends an appropriate line to the parent directory's checkout comment.

The **mkdir** command creates one or more *directory elements*. (You can also use the standard UNIX **mkdir(1)** command, but that creates *view-private directories*, not elements.) Unless you specify the **–nco** (no checkout) option, the new directory is checked out automatically. A directory element must be checked out before you can create elements and VOB links within it.

The **mkelem –eltype directory** command is equivalent to this command.

The new directory element is associated with the same storage pools (source, derived object, and cleartext) as its parent directory element. You can assign the directory to different pools with the **chpool** command. Note that the directory itself is stored in the database, but files created in the directory are stored in the pools associated with the directory.

In a *snapshot view*, this command also updates the directory element.

Access Mode

In standard UNIX fashion, new directory elements are created with mode 777, as modified by your umask. The meanings of the read, write, and execute access permissions do not have their standard UNIX meanings. See the **protect** reference page for details.

Converting View-Private Directories

You cannot create a directory element with the same name as an existing view-private file or directory, and you cannot use **mkdir** to convert an existing view-private directory structure into

directory and file elements. To accomplish this task, use the **clearexport_ffile** and **clearimport** utilities.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. An error occurs if any of the following objects are locked: VOB, element type.

OPTIONS AND ARGUMENTS

CHECKOUT OF THE NEW DIRECTORY. *Default:* **mkdir** checks out the new directory element.

–nco
Suppresses checkout of the new directory element.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

NAMING THE DIRECTORIES. *Default:* None.

dir-pname ...
One or more pathnames, specifying directories to be created.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a subdirectory named **subd**, and check out the directory to the current view.

  cmd-context **mkdir –nc subd**
  ```
  Created directory element "subd".
  Checked out "subd" from version "/main/0".
  ```

- Create a subdirectory named **release**, but do not check it out. Provide a comment on the command line.

  cmd-context **mkdir -nco -c "Storage directory for released files" release**
  ```
  Created directory element "release".
  ```

SEE ALSO

checkout, **mv**, **protect**, **pwd**, **rmelem**, **update**

mkelem

Creates a file or directory element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkelem [**–elt**·**ype** *element-type-name*] [**–nco** | **–ci** [**–pti**·**me**]] [**–master**] [**–nwa**·**rn**]
[**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* | **–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
element-pname ...

DESCRIPTION

The **mkelem** command creates one or more new *elements*. A new element can be created in a directory only if that directory is checked out. **mkelem** appends an appropriate line to the directory's checkout comment.

In Attache, any corresponding local files are uploaded before the command is executed remotely. Wildcards are expanded locally by searching in the workspace, rather than remotely.

mkelem processes each element as follows:

1. Determines an *element type* from the specified **–eltype** option or by performing file-typing (see *File Types and Element Types*)
2. Creates an element object with that element type in the appropriate VOB database
3. If you are using the **–ci** option to convert a view-private file to an element, uses the permissions of that file including **set-UID** and/or **set-GID** bits; otherwise, sets the access mode of the new element to `444` (file element) or `777` (directory element), as modified by your current **umask(1)** setting
4. Initializes the element's version tree by creating a single branch (named **main**), and a single, empty version (version 0) on that branch
5. Does one of the following:
   - By default, checks out the element to your view.

     NOTE: At this point, other views see an empty file when they look at the element.
   - With the **–nco** option, does nothing.

- With the **–ci** option, creates version 1 by copying a view-private file or an uploaded view-private file.

In Attache, if elements are checked out, the corresponding files are downloaded to your workspace if they did not exist locally, or the local files are made writable.

6. Assigns the element to the same *source storage pool*, *cleartext storage pool*, and (for new directory elements only) *derived object storage pool* as its parent directory element
7. In a *snapshot view*, updates the newly created element

NOTE: Error messages appear if your config spec lacks a **/main/LATEST** rule. The **mkelem** command succeeds in creating version **/main/0**. However, because your view does not have a rule to select this version, you cannot see or check out the element.

The following sections provide more information on each of these steps.

File Types and Element Types

Each element is an instance of an element type (just as each version label is an instance of a label type, each attribute is an instance of an attribute type, and so on). You can specify an element type with the **–eltype** option. (The **lstype –kind eltype** command lists a VOB's element types.) The element type must already exist in the VOB in which you are creating the new element, or must exist in the Admin VOB hierarchy associated with the VOB in which you are creating the new element. A **mkelem –eltype directory** command is equivalent to a **mkdir** command.

If you do not specify an element type on the command line, **mkelem** determines one by using the *magic files* to perform *file-typing*. This involves matching patterns in the new element's name (and examining the existing view-private file with that name, if one exists; see the section *Converting View-Private or Workspace Files to Elements*). If file-typing fails, an error occurs and no element is created:

```
cleartool: Error: Can't pick element type from rules in ...
```

For more on file-typing, see the **cc.magic** reference page.

Access Mode

Standard UNIX procedure is to create files with mode `666`, and directories with mode `777`, as modified by your umask. But a file element's write access settings are essentially irrelevant; modifications to elements are controlled by ClearCase or ClearCase LT permissions, as described in the **permissions** reference page. When your view selects a checked-in version of a file element, all of its write permissions are turned off, corresponding to the fact that the element is read-only. When you check out an element, write permissions are added to the view-private copy. (See the **checkout** reference page for details.)

In view of the fundamental read-only status of file elements, the mode to which your umask is applied is `444` (not `666`) for a file element.When you convert a view-private file to an element (see

Converting View-Private or Workspace Files to Elements), its read and execute rights become those of the new element. .

Converting View-Private or Workspace Files to Elements

You can use the **mkelem** command to convert a view-private or workspace local file to a file element with the same name. There are several cases:

- By default, **mkelem** creates an empty version 0 of the new element, then checks out the new element to your view. The view-private file becomes the checked-out version of the new element.
- If you use the **–ci** option (check in), **mkelem** does all of the above, then proceeds to check in version 1 of the new element. Thus, version 1 has the contents of the view-private file. With **–ptime**, **mkelem** preserves the modification time of the file being checked in.
- If you use the **–nco** option (no check out), **mkelem** just creates an empty version 0 of the new element.
- (Attache only) If the file exists locally, it is uploaded to the view.
- (Attache only) If you do not use the **–nco** option and the file did not exist locally, it is downloaded to your workspace. If the file exists locally, it is made writable.

During the element-creation process, **mkelem** renames the view-private or uploaded view-private file to prevent a name collision that would affect other ClearCase, ClearCase LT, or Attache software (for example, triggers on the **mkelem** operation). If this renaming fails, a warning message appears. There are two renaming procedures:

- If a **checkout** is performed on the new element, **mkelem** temporarily renames the view-private or uploaded view-private file, using a **.mkelem** (or possibly, **.mkelem**.*n*) suffix. After the new element has been created and checked out, **mkelem** restores the original name. This produces the intended effect: the data formerly in a view-private file is now accessible through an element with the same name.
- If no checkout is performed on the new element, **mkelem** alerts you that the view-private or uploaded view-private file has been renamed, using a **.keep** (or possibly, **.keep.***n*) extension. Note that the view-private file has not really been converted to an element — it has been moved out of the way to allow creation of an element in its place.

NOTE: If **mkelem** does not complete correctly, your view-private file or uploaded view-private file may be left under the **.mkelem** file name. In Attache, the file in your workspace is not renamed.

Converting Nonshareable Derived Objects to Elements

mkelem does not perform any special processing for a *nonshareable DO*. The process is the same as for a *shareable DO*, as described in *Converting View-Private or Workspace Files to Elements*.

However, when you check in version 1 of the new element (with the **–ci** option or the **checkin** command), the command converts the nonshareable DO to a shareable DO, then checks it in. For more information, see *Working with Derived Objects and Configuration Records* in *Building Software with ClearCase*.

NOTE: When a nonshareable DO is converted to a shareable DO, its DO-ID changes. For more information, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*.

Creating Directory Elements

To create a new directory element, use **mkelem –eltype directory** or **mkdir**. You cannot create a directory element with the same name as an existing view-private file or directory, and you cannot use **mkelem** to convert an existing view-private directory structure into directory and file elements. To accomplish this task, use the **clearexport_ffile** and **clearimport** utilities.

Auto-Make-Branch During Element Creation

If your config spec has a **/main/LATEST** rule with a **–mkbranch** clause, **mkelem** checks out a subbranch instead of the **main** branch. For example, suppose your view has this config spec when checked out:

```
element * CHECKEDOUT
element * .../gopher_port/LATEST
element * V1.0.1 -mkbranch gopher_port
element * /main/LATEST -mkbranch gopher_port
```

In this case, a **gopher_port** branchis created a for the new element, and this branch is checked out instead of **main**:

```
cmd-context  mkelem –c "new element for Gopher porting work" base.h
Created element "base.h" (type "text_file").
Created branch "gopher_port" from "base.h" version "/main/0".
Checked out "base.h" from version "/main/gopher_port/0".
```

The **auto-make-branch** facility is not invoked if you use the **–nco** option to suppress checkout of the new element. For more on this facility, see the **checkout** and **config_spec** reference pages.

Creating Elements in Replicated VOBs

By default, when you create an element in a replicated VOB, **mkelem** assigns mastership of the element's main branch to the VOB replica that masters the branch type **main**. If this replica is not your current replica, you cannot create versions on the main branch. (You can create versions on other branches if they are mastered by the current replica.)

To assign mastership of a new element's main branch to the current replica, use the **–master** option. The **–master** option also allows auto-make-branch during element creation, even if the branch type specified in your config spec is not mastered by the current replica. In this case, **mkelem** assigns mastership of newly created branches to the current replica. For example, suppose your view has the following config spec:

```
element * CHECKEDOUT
element * .../gms_dev/LATEST
element * /main/LATEST -mkbranch gms_dev
```

When you create a new element with **mkelem –master** and do not use the **–nco** option, **mkelem** creates the branches **main** and **gms_dev** and assigns their mastership to the current replica.

NOTE: If you use the **–nco** option with **–master**, only the main branch is mastered by the current replica, because it is the only branch created by **mkelem**.

Referencing Element Objects and Their Versions

You sometimes need to distinguish an element itself from the particular version of the element that is selected by your view. In general:

- Appending the extended naming symbol (by default, @@) to an element's name references the element itself.
- A simple name (no extended naming symbol) is a reference to the version in the view.

For example, **msg.c@@** references an element, whereas **msg.c** references a version of that element. In many contexts (for example, **checkin** and **lsvtree**), ClearCase and ClearCase LT allow you to ignore the distinction. But there are ambiguous contexts in which you need to be careful. For example, you can attach attributes and hyperlinks either to version objects or to element objects. Thus, these two commands are different:

cmd-context **mkattr BugNum 403 msg.c** *(attaches attribute to version)*

cmd-context **mkattr BugNum 403 msg.c@@** *(attaches attribute to element)*

CAUTION: Do not create elements whose names end with the extended naming symbol. ClearCase, ClearCase LT, and Attache cannot handle such elements.

Storage Pools

Physical storage for an element's versions (*data containers*) is allocated in the storage pools that **mkelem** assigns. You can change pool assignments with the **chpool** command.

(ClearCase and ClearCase LT only) Group Membership Restriction

Each VOB has a *group list*. You can create an element in a VOB only if your principal *group*—the first (or only) group listed when you enter an **id(1)** command—is on this list. See the **protectvob** reference page for more on this topic.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* An error occurs if any of the following objects are locked: VOB, element type, pool (nondirectory elements only).

OPTIONS AND ARGUMENTS

SPECIFYING THE ELEMENT TYPE. *Default:* **mkelem** performs file-typing to select an element type. If file-typing fails, an error occurs. See the **cc.magic** reference page for details on file-typing.

–elt·ype *element-type-name*
Specifies the type of element to be created. The element type must be a predefined type, or a user-defined type created with the **mkeltype** command. The element type must exist in each VOB in which you are creating a new element, or (if *element-type-selector* is a global type) in the Admin VOB hierarchy associated with each VOB. Specifying **–eltype directory** is equivalent to using the **mkdir** command.

CHECKOUT OF THE NEW ELEMENT. *Default:* **mkelem** checks out the new element. If a view-private file already exists at that pathname, it becomes the checked-out version of the element. Otherwise, an empty view-private file is created and becomes the checked-out version. In Attache, if neither the **–nco** or **–ci** option is specified, the checked-out files are downloaded if they did not exist locally, or the local files are made writable.

–nco
Suppresses automatic checkout; **mkelem** creates the new element, along with the **main** branch and version**/main/0**, but does not check it out. If *element-pname* exists, it is moved aside to a **.keep** file, as explained earlier.

–ci [**–pti·me**]
Creates the new element and version **/main/0**, performs a checkout, and checks in a new version containing the data in view-private file or DO *element-pname*, which must exist. In Attache, local files corresponding to successfully checked-in versions are made read-only. You cannot use this option when creating a directory element.

With **–ptime**, **mkelem** preserves the modification time of the file being checked in. If you omit this option, the modification time of the new version is set to the checkin time.

NOTE: On some platforms, it is important that the modification time be preserved for archive files (libraries) created by **ar(1)** (and perhaps updated with **ranlib(1)**). The link editor, **ld(1)**, will complain if the modification time does not match a time recorded in the archive itself. Be sure to use this option, or (more reliably) store archive files as elements of a user-defined type, created with the **mkeltype –ptime** command. This causes **–ptime** to be invoked when the element is checked in.

MASTERSHIP OF THE MAIN BRANCH. *Default:* Assigns mastership of the element's **main** branch to the VOB replica that masters the **main** branch type.

–master
Assigns mastership of the **main** branch of the element to the VOB replica in which you execute the **mkelem** command. If your config spec includes **–mkbranch** lines or **mkbranch** rules that apply to the element, and you do not use the **–nco** option, **mkelem**

creates these branches and assigns their mastership to the current VOB replica. **mkelem** also prints a note that these branches are explicitly mastered by the current replica; the output also displays the master replica of each associated branch type.

SUPPRESSING WARNING MESSAGES *Default:* Warning messages are displayed.

–nwa·rn
Suppresses warning messages.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE ELEMENTS. *Default:* None.

element-pname ...
The pathnames of one or more elements to be created. If you also specify the **–ci** option, each *element-pname* must name an existing view-private object. You cannot create a directory element with the same name as an existing view-private file or directory.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a file element named **rotate.c** of type **compressed_text_file**, and check out the initial version (version 0).

  cmd-context **mkelem –nc –eltype compressed_text_file rotate.c**
  ```
  Created element "rotate.c" (type "compressed_text_file").
  Checked out "rotate.c" from version "/main/0".
  ```

- Create three file elements, **cm_add.c**, **cm_fill.c**, and **msg.c**, allowing the file-typing mechanism to determine the element types. Do not check out the initial versions.

  cmd-context **mkelem –nc –nco cm_add.c cm_fill.c msg.c**
  ```
  Created element "cm_add.c" (type "text_file").
  Created element "cm_fill.c" (type "text_file").
  Created element "msg.c" (type "text_file").
  ```

- Convert a view-private file named **test_cmd.c**, to an element, and check in the initial version.

cmd-context **mkelem –nc –ci test_cmd.c**
```
Created element "test_cmd.c" (type "text_file").
Checked in "test_cmd.c" version "/main/1".
```

- Create two directory elements and check out the initial version of each.

  cmd-context **mkelem –nc –eltype directory libs include**
  ```
  Created element "libs" (type "directory").
  Checked out "libs" from version "/main/0".
  Created element "include" (type "directory").
  Checked out "include" from version "/main/0".
  ```

- Create an element type named **archive** for library archive files, with the predefined **binary_delta_file** as its supertype. Then, change to the **libs** directory, check it out, and create two elements of type **archive** without checking them out.

  cmd-context **mkeltype –nc –supertype binary_delta_file archive**
  ```
  Created element type "archive".
  ```

  cmd-context **cd libs**

  cmd-context **co –nc .**
  ```
  Checked out "." from version "/main/1".
  ```

  cmd-context **mkelem –nc –nco –eltype archive libntx.a libpvt.a**
  ```
  Created element "libntx.a" (type "archive").
  Created element "libpvt.a" (type "archive").
  ```

SEE ALSO

cc.magic, **checkin**, **checkout**, **chpool**, **config_spec**, **lstype**, **mkdir**, **mkeltype**, **mkpool**, **protect**, **update**

mkeltype

Creates or updates an element type object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkeltype [**–rep·lace**] [**–glo·bal** [**–acq·uire**] | **–ord·inary**]
–sup·ertype *elem-type-selector* [**–man·ager** *mgr-name*]
[**–pti·me**] [**–att·ype** *attr-type-selector*[,...]]
[–**mer·getype** { **auto** | **user** | **never** }]
[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
element-type-selector ...

DESCRIPTION

The **mkeltype** command creates one or more user-defined *element types* for future use within a VOB. User-defined element types are variants of the predefined types. (See complete list in the section *Predefined Element Types*.) After creating an element type, you can create elements of that type using **mkelem**, or change an existing element's type using **chtype**. To remove an element type, use the **rmtype** command.

NOTE: You cannot remove an element type from a replicated VOB.

Setting Merge Behavior for an Element Type

In some cases, you can select the merge behavior of an element type when you create it. This is true for element types of elements used in a UCM deliver or rebase operation. (See the **deliver** and **rebase** reference pages). There are three kinds of behaviors, described here with their associated keywords.

Keyword	Behavior
auto (default)	A ClearCase or ClearCase LT **findmerge** operation attempts to merge elements of this type.
user	A ClearCase or ClearCase LT **findmerge** operation performs trivial merges only. Nontrivial merges must be made manually.

Keyword	Behavior
never	A ClearCase or ClearCase LT **findmerge** operation ignores elements of this type. The **never** attribute is useful for working with files such as binary files or bitmap graphics images.

To specify a behavior, use one of the keywords as the argument to the –**mergetype** option. If the option is not specified, automatic merge behavior is in effect for elements of this element type.

Element Supertypes

When you create a new element type, you must specify an existing element type as its *supertype*. The new element type inherits the *type manager* of the supertype, unless you use the **–manager** option. The type manager performs such tasks as storing/retrieving the contents of the element's versions. (See the **type_manager** reference page.)

For example, you create an element type **c_source**, with **text_file** as the supertype; **c_source** inherits the type manager associated with the **text_file** supertype—the **text_file_delta** manager.

You can use the **lstype** command to list both the supertype and the type manager of an element type.

Predefined Element Types

Each VOB is created with the following element types:

file	Versions can contain any kind of data (text, binary, bitmap, and so on). Uses the **whole_copy** type manager.
compressed_file	Versions can contain any kind of data. Uses the **z_whole_copy** type manager.
text_file	All versions must contain text (multibyte text characters are allowed). Null bytes are not permitted (a byte of all zeros); no line can contain more than 8,000 characters. Uses the **text_file_delta** type manager.
compressed_text_file	All versions must contain text; no line can contain more than 8,000 characters. Uses the **z_text_file_delta** type manager.
binary_delta_file	Versions can contain any kind of data. Uses the **binary_delta** type manager.
html	Subtype of the **text_file** element type. Uses the **_html** type manager.
ms_word	Subtype of the **file** element type. All versions must be Microsoft Word files. Uses the **_ms_word** type manager.
rose	Subtype of the **text_file** element type. Uses the **_rose** type manager.
xml	Subtype of the **text_file** element type. Uses the **_xml** type manager.

directory	Versions of a directory element catalog (list the names of) elements and VOB symbolic links. Uses the **directory** type manager, which compares and merges versions of directory elements.
file_system_object	Generic element type, with no associated type manager.

You can use any of these element types as the **–supertype** specification.

Text Files, Cleartext, and a View's Text Mode

This section applies to the element types *text_file* and *compressed_text_file*, to all subtypes of these types, and to all user-defined element types derived from them through the supertype mechanism.

When a load operation is issued from a snapshot view, or a user program accesses a version through a *dynamic view*, the type manager handles it as follows:

1. Extracts the text lines of that particular version from the data container.
2. Stores the extracted lines in a *cleartext file*, within the cleartext storage pool directory associated with the element.
3. Arranges for the program to access the cleartext file (not the structured data container).

On subsequent accesses to the same version, steps 1 and 2 are skipped; the program accesses the existing cleartext file, which is cached in the cleartext storage pool.

Operating systems vary in their use of text-file line terminators. To avoid confusion, each ClearCase and ClearCase LT view has a *text mode*, which determines the line terminator for text files in that view. (See the **mkview** reference page.) After the type manager constructs a cleartext file for a version, its line terminators may be adjusted before the version is presented to the calling program. Adjustment of line terminators can also occur when the **checkout** command copies a version of a text file element, creating a view-private file (the checked-out version).

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following (with **–replace** only): type owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type (with **–replace** only).

OPTIONS AND ARGUMENTS

HANDLING OF NAME COLLISIONS. *Default:* An error occurs if an element type named *type-name* already exists in the VOB.

–rep·lace
Replaces the existing definition of *type-name* with a new one. If you do not include options from the existing definition, their values are replaced with the defaults.

(Exception: the type's scope does not change unless you explicitly specify a **–global** or **–ordinary** option.)

If you specify a comment when using **–replace**, the comment appears in the event record for the modification (displayed with **lshistory –minor**); it will not replace the object's creation comment (displayed with **describe**). To change an object's creation comment, use **chevent**.

You cannot change the following:

- The type manager (**–manager** or **–supertype** option) if there are existing elements of type *type-name*
- The definition of a predefined element type (such as **file** or **text_file**)

Also, when converting a global type to ordinary, you must specify the global type as the *element-type-selector* argument. You cannot specify a local copy of the global type.

SPECIFYING THE SCOPE OF THE ELEMENT TYPE. *Default:* Creates an ordinary element type that can be used only in the current VOB.

–glo·bal [**–acq·uire**]
Creates an element type that can be used as a global resource by client VOBs in the administrative VOB hierarchy. With **–acquire**, **mkeltype** checks all eclipsing types in client VOBs and converts them to local copies of the new global type.

For more information, see *Administering ClearCase*.

–ord·inary
Creates an element type that can be used only in the current VOB.

SUPERTYPE / TYPE MANAGER INHERITANCE. *Default:* None. You must specify a supertype; the new element type inherits the type manager of this supertype, unless you use the **–manager** option.

–sup·ertype *elem-type-selector*
The name of an existing element type, predefined or user-defined. Predefined element types are listed in *Predefined Element Types* on page 601. You can specify **–supertype file_system_object** only if you also specify a type manager with **–manager**.

Specify *element-type-selector* in the form [**eltype:**]*type-name*[**@***vob-selector*]

type-name	Name of the element type See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

The **lstype** command lists a VOB's existing element types.

–man·ager *mgr-name*
Specifies the type manager for the new element type, overriding inheritance from the supertype. *Predefined Element Types* on page 601 lists the type managers. For more information about these type managers, see the **type_manager** reference page.

CONTROLLING VERSION-CREATION TIME. *Default:* For all elements of the newly created type: when a new version is checked in, its time stamp is set to the checkin time.

–pti·me
For all elements of the newly created type: preserves the time stamp of the checked-out version during checkin. In effect, this establishes **checkin –ptime** as the default for elements of this type.

MERGETYPE. *Default:* Instantiations of the new element type use automatic merging**.**

–mer·getype *keyword*
Specifies the merge behavior for an element type. This is in effect only when the element type is used in a UCM deliver or rebase operation. There are three types of merge behavior: automatic, for which a **findmerge** operation attempts to automatically merge elements; user-controlled, for which a **findmerge** operation performs trivial merges only (other merges must be made manually); and never, meaning **findmerge** ignores elements of this type. The corresponding keyword arguments are **auto**, **user**, and **never; auto** is the default.

SUGGESTED ATTRIBUTES. *Default:* The new element type has no list of suggested attributes.

–att·ype *attr-type-selector*[,...]
A comma-separated list (no white space) of existing attribute types. Use this option to inform users of suggested attributes for use with elements of the newly created type. This does not restrict users from using other attributes. (Users can view the list with **describe** or **lstype**.) Specify *attribute-type-selector* in the form [**attype:**]*type-name*[**@***vob-selector*]

type-name — Name of the attribute type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

NAMING THE ELEMENT TYPES. *Default:* The element type is created in the VOB that contains the current working directory unless you specify another VOB with the **@***vob-selector* argument.

type-name ...
Names of the element types to be created. Specify *element-type-selector* in the form [**eltype:**]*type-name*[**@***vob-selector*]

type-name — Name of the element type
See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create an element type named **c_source** using the predefined **text_file** element type as the supertype.

  cmd-context **mkeltype –supertype text_file –nc c_source**
  ```
  Created element type "c_source".
  ```

- Create an element type for storing binary data named **bin_file**, using the predefined **file** element type as the supertype.

  cmd-context **mkeltype –supertype file –nc bin_file**
  ```
  Created element type "bin_file".
  ```

- Create an element type based on the user-defined element type **bin_file** (from previous example) for storing executable files. Include an attribute list.

  cmd-context **mkeltype –supertype bin_file –attype Confidence_Level,QAed –nc exe_file**
  ```
  Created element type "exe_file".
  ```

- Create a "directory of include files" element type, using the predefined **directory** element type as the supertype. Provide a comment on the command line.

  cmd-context **mkeltype –supertype directory –c "directory type for include files" incl_dir**
  ```
  Created element type "incl_dir".
  ```

- Change the **checkin** default for an existing element type so that it preserves the file modification time. Provide a comment on the command line.

  cmd-context **mkeltype –replace –supertype bin_file –ptime **
  –c "change archive mod time default" archive
  ```
  Replaced definition of element type "archive".
  ```

- Create an element type for storing binary data named **grph_file**, using the predefined **file** element type as the supertype. Specify the merge type as never. Merge type information is applied when an element of this type is used in a UCM deliver or rebase operation.

  cmd-context **mkeltype –supertype file –mergetype never –nc grph_file**
  ```
  Created element type "grph_file".
  ```

SEE ALSO

checkin, **chtype**, **describe**, **lstype**, **mkelem**, **rmtype**, **rename**, **type_manager**, **type_object**

mkfolder

Creates a folder for a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

mkfolder [**–c**·**omment** *comment* | **–cfi**·**le** *pname* | **–cq**·**uery** | **–cqe ach** | **–nc omment**]
[**–title** *title*] **–in** *parent-folder-selector* [*folder-selector* **...**]

DESCRIPTION

The **mkfolder** command creates a folder for a UCM project. Folders have these characteristics:

- They can contain projects or other folders.
- They must reside in a UCM project VOB.
- Each folder must have a parent folder.

The parent folder for a top-level folder is named **RootFolder**, a predefined object.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions are required.

Locks: An error occurs if there are locks on any of the following objects: UCM project VOB.

Mastership: There are no mastership requirements.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–c**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE FOLDER TITLE. *Default*: The folder's name, specified as part of the *folder-selector* argument.

–title *title*
Specifies a descriptive title displayed in output and the graphical interface for all folders

created. The *title* argument can be a character string of any length. Use double quotes to enclose titles with spaces or special characters.

SPECIFYING THE PARENT FOLDER. *Default*: None.

–in *parent-folder-selector*
Specifies a parent folder for the new folder. To create a top-level folder, you must specify the predefined folder object **RootFolder** as its parent folder.

folder-selector is of the form: [**folder:**]*folder-name*[*@vob-selector*] and *vob* is the folder's UCM project VOB.

SPECIFYING THE FOLDER NAME. *Default*: A generated name.

folder-selector **...**
Identifies one or more new folders.Each folder must reside in the same UCM project VOB as its parent folder and is created in the folder specified by the –**in** option.

folder-selector is of the form: [**folder:**]*folder-name*[*@vob-selector*] and *vob* is the folder's UCM project VOB.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Creates a top-level folder whose parent is the predefined object **RootFolder**.

```
cmd-context    mkfolder -title "webo projects" -in \
RootFolder@/vobs/webo_pvob webo_projects@/vobs/webo_pvob

Created folder "webo_projects".
```

SEE ALSO

chfolder, **lsfolder**, **mkproject**, **rmfolder**

mkhlink

Attaches a hyperlink to an object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkhlink [**–uni·dir**] [**–tte·xt** *to-text*] [**–fte·xt** *from-text*]
[**–fpn·ame**] [**–tpn·ame**]
[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
hlink-type-selector from-obj-selector [*to-obj-selector*]

DESCRIPTION

The **mkhlink** command creates a *hyperlink* between two objects, each of which may be an *element*, *branch*, *version*, *VOB symbolic link*, or non-file-system VOB object (except another hyperlink).

Logically, a hyperlink is an "arrow" attached to one or two VOB-database objects:

- A *bidirectional* hyperlink connects two objects, in the same VOB or in different VOBs, with optional text annotations at either end. It can be navigated in either direction: from-object → to-object or to-object → from-object.
- A *unidirectional* hyperlink connects two objects in different VOBs, with optional text annotations at either end. It can be navigated only in the from-object → to-object direction.
- A *text-only* hyperlink associates one object with a user-defined text string (for example, an element that implements a particular algorithm with the name of a document that describes it).
- A *null-ended* hyperlink has only a from-object. Use this kind of hyperlink to suppress *hyperlink inheritance* (see the *Hyperlink Inheritance* section).

Contrast with Other Kinds of Metadata

Like other kinds of metadata annotations—version labels, attributes, and triggers—a hyperlink is an instance of a type object: the **mkhlink** command creates an instance of an existing **hyperlink type** object. However, hyperlinks differ from other kinds of metadata annotations:

- The hyperlink created by **mkhlink** is also an object in itself. Each hyperlink object has a unique ID (see the *Hyperlink-IDs* section) and can itself be annotated with attributes. By contrast, a **mklabel**, **mkattr**, or **mktrigger** command does not create a new object; it simply annotates an existing object.
- You can attach several hyperlinks of the same type to one object, but only one instance of a particular label, attribute, or trigger type. (For example, you can attach two different **DesignFor** hyperlinks to the same object, but not two different **ECOnum** attributes.)

Hyperlink-IDs

Each new hyperlink object gets a unique identifier, its *hyperlink-ID*. You can specify any hyperlink by appending its hyperlink-ID to the name of the hyperlink type. Following are some examples.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

cmd-context **describe hlink:DesignFor@52179@/vobs/doctn**

In this example, **DesignFor** is the name of a hyperlink type, and **@52179@/vobs/doctn** is the hyperlink-ID. Note that the hyperlink-ID includes a pathname: the VOB-tag of the VOB in which the hyperlink is created. When specifying a hyperlink, you can use any pathname within the VOB in place of the VOB-tag pathname:

cd /vobs

cmd-context **describe hlink:DesignFor@52179@doctn**

You can omit the pathname if the current working directory is in that VOB:

cd /vobs/doctn

cmd-context **describe hlink:DesignFor@52179**

A hyperlink-ID is unique in that it is guaranteed to differ from the hyperlink-ID of all other hyperlinks. But it can change over time; when a VOB's database is processed with **reformatvob**, the numeric suffixes of all hyperlink-IDs change:

```
before 'reformatvob':@52179@/vobs/doctn
after 'reformatvob':@8883@/vobs/doctn
```

Similarly, the VOB-tag part of a hyperlink-ID can change over time and can vary from host to host.

Hyperlink Inheritance

By default, a version implicitly inherits a hyperlink attached to any of its ancestor versions, on the same branch or on a parent branch. Inherited hyperlinks are listed by the **describe** command only if you use the **–ihlink** option.

A hyperlink stops being passed down to its descendents if it is superseded by another hyperlink of the same type, explicitly attached to some descendent version. You can use a *null-ended* hyperlink (from-object, but no to-object) as the superseding hyperlink to effectively cancel hyperlink inheritance.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, object group member, object owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type, object or object type (for non-file-system objects).

OPTIONS AND ARGUMENTS

UNIDIRECTIONAL/BIDIRECTIONAL. *Default:* Creates a *bidirectional* hyperlink. If the objects being linked are in different VOBs, a notation is made in the VOB database of the to-object, making it possible to see the hyperlink from either VOB.

–uni·dir
Creates a *unidirectional* hyperlink; no notation is made in the VOB database of the to-object (if that object is in a different VOB).

NOTE: In all cases, a single hyperlink object is created, in the VOB of the from-object.

TEXT ANNOTATIONS. *Default:* The hyperlink has no text annotations.

–tte·xt *to-text*
Text associated with the to-end of a hyperlink. If you also specify *to-obj-pname*, the text is associated with that object. If you do not specify *to-obj-pname*, **cleartool** creates a *text-only* hyperlink, originating from *from-obj-pname*. If you omit both **–ttext** and *to-obj-pname*, **cleartool** creates a *null-ended* hyperlink.

–fte·xt *from-text*
Text associated with the from-end of a hyperlink.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE HYPERLINK TYPE. *Default:* None.

hlink-type-selector

An existing hyperlink type. The hyperlink type must exist in each VOB containing an object to be hyperlinked, or (if *hlink-type-selector* is a global type) in the Admin VOB hierarchy associated with each VOB. Specify *hlink-type-selector* in the form [**hltype:**]*type-name*[**@***vob-selector*]

type-name	Name of the hyperlink type
vob-selector	Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

OBJECTS TO BE HYPERLINKED. *Default:* None. You must specify at least one object.

[**–fpn**·**ame**] *from-obj-selector*
[**–tpn**·**ame**] *to-obj-selector*

from-obj-selector specifies the from-object, and *to-obj-selector* specifies the to-object. *to-obj-selector* is optional; omitting it creates a text-only hyperlink (if you use **–ttext**) or a null-ended hyperlink (if you don't).

NOTE: An error occurs if you try to make a unidirectional hyperlink whose *to-obj-selector* is a checked-out version in another VOB.

Specify *from-obj-selector* and *to-obj-selector* in one of the following forms:

pname

- A standard or view-extended pathname to an element specifies the version in the view.
- A version-extended pathname specifies an element, branch, or version, independent of view.
- The pathname of a VOB symbolic link.

NOTE: If *pname* has the form of an object selector, you must include the **–fpname** or **–tpname** option to indicate that *pname* is a pathname.

Examples:

`foo.c`	*(version of 'foo.c' selected by current view)*
`/view/gamma/usr/project/src/foo.c`	*(version of 'foo.c' selected by another view)*
`foo.c@@/main/5`	*(version 5 on main branch of 'foo.c')*
`foo.c@@/REL3`	*(version of 'foo.c' with version label 'REL3')*
`foo.c@@`	*(the element 'foo.c')*
`foo.c@@/main`	*(the main branch of element 'foo.c')*

vob-selector **vob:***pname-in-vob*

pname-in-vob can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any filesystem object within the VOB (if the VOB is mounted). It cannot be the pathname of the *VOB storage directory.*

attribute-type-selector	**attype:***type-name*[**@***vob-selector*]
branch-type-selector	**brtype:***type-name*[**@***vob-selector*]
element-type-selector	**eltype:***type-name*[**@***vob-selector*]
hyperlink-type-selector	**hltype:***type-name*[**@***vob-selector*]
label-type-selector	**lbtype:***type-name*[**@***vob-selector*]
trigger-type-selector	**trtype:***type-name*[**@***vob-selector*]
pool-selector	**pool:***pool-name*[**@***vob-selector*]
oid-obj-selector	**oid:***object-oid*[**@***vob-selector*]

The following object selector is valid only if you use MultiSite:

replica-selector	**replica:***replica-name*[**@***vob-selector*]

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a hyperlink type. Then create a unidirectional, element-to-element hyperlink between an executable and its GUI counterpart in another VOB.

cmd-context **mkhltype -nc gui_tool**
```
Created hyperlink type "gui_tool".
```

cmd-context **mkhlink -unidir gui_tool monet@@ /vobs/gui/bin/xmonet@@**
```
Created hyperlink "gui_tool@1239@/usr/hw".
```

- Create a hyperlink of type **design_spec** connecting the versions of a source file and design document labeled **REL2**.

  cmd-context **mkhlink design_spec util.c@@/REL2 /usr/hw/doc/util.doc@@/REL2**
  ```
  Created hyperlink "design_spec@685@/usr/hw".
  ```

- Create three hyperlinks of the same type from the same version of a design document; each hyperlink points to a different source file element.

  cmd-context **mkhlink design_for sortmerge.doc ../src/sort.c**
  ```
  Created hyperlink "design_for@4249@/vobs/proj".
  ```

  cmd-context **mkhlink design_for sortmerge.doc ../src/merge.c**
  ```
  Created hyperlink "design_for@4254@/vobs/proj".
  ```

  cmd-context **mkhlink design_for sortmerge.doc ../src/sortmerge.h**
  ```
  Created hyperlink "design_for@4261@/vobs/proj".
  ```

- Create an element-to-element hyperlink between a source file and a script that tests it. Specify both from-text and to-text for further annotation.

  cmd-context **mkhlink -ttext "regression A" -ftext "edge effects" tested_by **
  cm_add.c@@ edge.sh@@
  ```
  Created hyperlink "tested_by@714@/usr/hw".
  ```

- Create a hyperlink of type **fixes** between the version of **util.c** in your view and the element **bug.report.21**. Use to-text to indicate the bug number ("fixes bug 21").

  cmd-context **mkhlink -ttext "fixes bug 21" fixes util.c /usr/hw/bugs/bug.report.21@@**
  ```
  Created hyperlink "fixes@746@/usr/hw".
  ```

- Create a **text only** hyperlink of type **design_spec** to associate the algorithm **convolution.c** with the third-party document describing that algorithm. Make the hyperlink between the element **convolution.c** and the to-text that describes it.

  cmd-context **mkhlink -ttext "Wilson: Digital Filtering, p42-50" **
  design_spec convolution.c@@
  ```
  Created hyperlink "design_spec@753@/usr/hw".
  ```

SEE ALSO

describe, **lstype**, **mkhltype**, **rename**, **rmhlink**, **xclearcase**

mkhltype

Creates or updates a hyperlink type object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mkhltype [**–rep**·**lace**] [**–glo**·**bal** [**–acq**·**uire**] | **–ord**·**inary**]
[**–att**·**ype** *attr-type-selector*[,...]] [**–sha**·**red**]
[**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
hlink-type-selector ...

DESCRIPTION

Th **mkhltype** command creates one or more *hyperlink types* for future use within a VOB. After creating a hyperlink type, you can connect pairs of objects with hyperlinks of that type, using **mkhlink**.

Conceptually, a *hyperlink* is an "arrow" from one *VOB-database* object (version, branch, element, or VOB symbolic link) to another. To enable objects in two different VOBs to be connected, a hyperlink type with the same name must be created in both VOBs.

For example, you create a hyperlink type named **design_spec**, for use in linking source code files to the associated design documents. Later, you can use **mkhlink** to create a hyperlink of this type between **my_prog.c** and **my_prog.dsn**.

Predefined Hyperlink Types

The following predefined hyperlink types are created in a new VOB:

Merge — **Merge** hyperlinks record a merge of two or more versions of an element (performed by the **merge** command) with one or more merge arrows. Each merge arrow is actually a hyperlink of type **Merge**, connecting one of the contributors to the target version.

GlobalDefinition — **GlobalDefinition** hyperlinks record the relationship between a global definition and a local instance of a global type. (See the **type_object** reference page.)

AdminVOB	**AdminVOB** hyperlinks record a VOB's administration VOB.
RelocationVOB	**RelocationVOB** hyperlinks point from VOBs to which objects have been relocated to the VOBs in which the objects were originally located. These hyperlinks occur only between VOB objects. (See the **relocate** reference page.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following (with **–replace** only): type owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, hyperlink type (with **–replace** only).

OPTIONS AND ARGUMENTS

HANDLING OF NAME COLLISIONS. *Default:* An error occurs if a hyperlink type named *type-name* already exists in the VOB.

–rep·lace

Replaces the existing definition of *type-name* with a new one. If you do not include options from the existing definition, their values are replaced with the defaults (Exception: the type's global scope does not change; you must explicitly specify **–global** or **–ordinary**).

If you specify a comment when using **–replace**, the comment appears in the event record for the modification (displayed with **lshistory –minor**); it does not replace the object's creation comment (displayed with **describe**). To change an object's creation comment, use **chevent**.

Constraints:

- You cannot replace predefined hyperlink types.
- When replacing a hyperlink type that was created with the **–shared** option, you must use **–shared** again; that is, you cannot convert a hyperlink type from shared to unshared.
- When converting a global type to ordinary, you must specify the global type as the *hlink-type-selector* argument. You cannot specify a local copy of the global type.

SPECIFYING THE SCOPE OF THE HYPERLINK TYPE. *Default:* Creates an ordinary hyperlink type that can be used only in the current VOB.

–glo·bal [**–acq·uire**]

Creates a hyperlink type that can be used as a global resource by client VOBs in the administrative VOB hierarchy. With **–acquire**, **mkhltype** checks all eclipsing types in client VOBs and converts them to local copies of the new global type.

For more information, see *Administering ClearCase*.

–ord·inary
Creates a hyperlink type that can be used only in the current VOB.

SUGGESTED ATTRIBUTES. (Advisory only, not restrictive) *Default:* The new hyperlink type has no list of suggested attributes.

–att·ype *attr-type-selector*[,...]
A comma-separated list (no white space) of existing attribute types. Use this option to inform users of suggested attributes for use with hyperlinks of the newly created type. (Users can view the list with **describe** or **lstype**.) See the **mkattype** and **mkattr** reference pages for more information about attributes.

MASTERSHIP OF THE HYPERLINK TYPE. *Default:* Attempts to attach hyperlinks of this type succeed only in the *VOB replica* that is the current *master* of the hyperlink type. The VOB replica in which the new hyperlink type is created becomes its initial master.

–sha·red
Hyperlinks of this type can be created in any VOB replica. (You can delete a hyperlink of this type only at the master site.)

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

NAMING THE HYPERLINK TYPES. *Default:* The hyperlink type is created in the VOB that contains the current working directory unless you specify another VOB with the **@***vob-selector* argument.

hlink-type-selector ...
Names of the hyperlink types to be created. Specify *hlink-type-selector* in the form [**hltype:**]*type-name*[**@***vob-selector*]

type-name	Name of the hyperlink type See the *Object Names* section in the **cleartool** reference page for rules about composing names.
vob-selector	Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a hyperlink type named **tested_by**.

cmd-context **mkhltype -nc tested_by**

```
Created hyperlink type "tested_by".
```

- Create a hyperlink type named **design_spec** in the **/vobs/docs** VOB, and provide a comment on the command line.

  cmd-context **mkhltype -c "source to design document" design_spec@/vobs/docs**

  ```
  Created hyperlink type "design_spec".
  ```

- Create a hyperlink type named **test_script**, providing a suggested-attribute list.

  cmd-context **mkhltype -nc -attype run_overnight,error_rate test_script**

  ```
  Created hyperlink type "test_script".
  ```

SEE ALSO

describe, **lstype**, **mkhlink**, **rename**, **rmtype**, **type_object**

mklabel

Attaches version labels to versions of elements

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- Attach label to specified versions:

  mklabel [**–rep**·**lace**] [**–r**·**ecurse**] [**–ver**·**sion** *version-selector*]
  [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
  label-type-selector pname ...

- Attach label to versions listed in configuration record:

  mklabel [**–rep**·**lace**] [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery**
  | **–cqe**·**ach** | **–nc**·**omment**]
  [**–sel**·**ect** *do-leaf-pattern*] [**–ci**] [**–typ**·**e** { **f** | **d** } ...]
  [**–nam**·**e** *tail-pattern*] **–con**·**fig** *do-pname label-type-selector*

DESCRIPTION

The **mklabel** command attaches a *version label* to one or more *versions*. You can attach a label to only one version of a particular element. You can specify the versions themselves on the command line, or you can specify a particular *derived object*. In the latter case, **mklabel** labels some or all the versions that were used to build that derived object.

Referencing Labeled Versions

Labeling a version of an element can affect the way the element appears in *views*. It also provides a new way to access the version with a version-extended pathname.

Version Selection by Views. A typical config spec rule uses version labels to select versions:

```
element * BASELEVEL_1
```

If you attach version label **BASELEVEL_1** to a version of element **foo.c**, any view configured with this rule selects the labeled version (unless some rule earlier in the config spec matches another version of **foo.c**).

Version Labels in Version-Extended Pathnames. Labeling a version effectively adds a new *hard link* to the version in the extended namespace. If you attach version label **R4.1A** to version **/main/rls4/12** of element **bar.c**, these pathnames are equivalent:

```
bar.c@@/main/rls4/12
bar.c@@/main/rls4/R4.1A
```

In addition, a third pathname is *usually* equivalent:

```
bar.c@@/R4.1A
```

This version-extended pathname is valid if it is unambiguous, that is, if no other version of **bar.c** is currently labeled **R4.1A**. (This is usually the case because, by default, label types are restricted to being used once per element. See the description of the **–pbranch** option in the **mklbtype** reference page.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, label type.

OPTIONS AND ARGUMENTS

MOVING A VERSION LABEL. *Default:* An error occurs if a version label of this type is already attached to some other version of the same element.

–rep·lace
Removes an existing label of the same type from another version of the element:

- From another version on the same branch, if *label-type-name* was created with **mklbtype –pbranch**
- From another version anywhere in the element's version tree, if *label-type-name* was not created with **mklbtype –pbranch**

No error occurs if there is no such label to remove, but the label is attached to all versions specified in the command.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE LABEL TYPE. *Default:* None.

label-type-selector

A label type, previously created with **mklbtype**. The label type must exist in each VOB containing a version to be labeled, or (if *label-type-selector* is a global type) in the Admin VOB hierarchy associated with each VOB. Specify *label-type-selector* in the form [**lbtype:**]*type-name*[@*vob-selector*]

type-name — Name of the label type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

DIRECTLY SPECIFYING THE VERSIONS TO BE LABELED. The options and arguments in this section specify elements and their versions directly on the command line. Do not use these options and arguments when using a derived object to provide a list of versions.

pname ...

(Required) One or more pathnames, indicating versions to be labeled:

- A standard or view-extended pathname to an element specifies the version selected in the view.
- A version-extended pathname specifies a version, independent of view.

Use **–version** to override these interpretations of *pname*.

NOTE: **mklabel** differs from some other commands in its default handling of directory element *pname* arguments: it labels the directory element itself; it does not label the elements cataloged in the directory (unless you specify **-recurse**).

–ver·sion *version-selector*

For each *pname*, attaches the label to the version specified by *version-selector*. This option overrides both version-selection and version-extended naming. See the **version_selector** reference page for syntax details.

–r·ecurse

Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are *not* traversed during the recursive descent into the subtree.

USING A DERIVED OBJECT TO SPECIFY THE VERSIONS TO BE LABELED. The options and arguments in this section specify versions by selecting them from the configuration records associated with

a particular derived object. Do not use these options when specifying elements and versions directly on the command line.

NOTE: Derived objects are created only in dynamic views.

–con·fig *do-pname*
(Required) Specifies one derived object. A standard pathname or view-extended pathname specifies the DO that currently appears in a view. To specify a DO independent of view, use an extended name that includes a *DO-ID* (for example, **hello.obj@@24–Mar.11:32.412**) or a version-extended pathname to a DO version.

With the exception of checked-out versions, **mklabel** labels all the versions that would be included in a **catcr –long –flat –element_only** listing of that derived object. Note that this includes the following objects:

- Any DO created by the build and subsequently checked in as a DO version.
- Any file in the CR that was view-private at the time of the build, was converted to an element after the creation of the CR, and has at least one checked-in version.

If the DO's configuration includes multiple versions of the same element, only the most recent version is labeled.

Use the following options to modify the list of versions to be labeled.

–sel·ect *do-leaf-pattern*
–ci
–typ·e { **f** | **d** } ...
–nam·e *tail-pattern*
Modify the set of versions to be labeled in the same way that these options modify a **catcr** listing. See the **catcr** reference page for details, and also the *EXAMPLES* section.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a label type named **REL6**. Attach that label to the version of the current directory selected by your view, and to the currently selected version of each element in and below the current directory.

  cmd-context **mklbtype –nc REL6**
  ```
  Created label type "REL6".
  ```

  cmd-context **mklabel –recurse REL6 .**

```
Created label "REL6" on "." version "/main/4".
Created label "REL6" on "./bin" version "/main/1".
Created label "REL6" on "./include" version "/main/1".
Created label "REL6" on "./libs" version "/main/2".
Created label "REL6" on "./lost+found" version "/main/0".
Created label "REL6" on "./release" version "/main/1".
Created label "REL6" on "./src" version "/main/6".
Created label "REL6" on "./src/Makefile" version "/main/2".
Created label "REL6" on "./src/cm_add.c" version "/main/1".
Created label "REL6" on "./src/convolution.c" version "/main/4".
Created label "REL6" on "./src/edge.sh" version "/main/1".
.
.
.
```

- Attach label **REL1** to the version of **msg.c** in the view.

  cmd-context **mklabel REL1 msg.c**

  ```
  Created label "REL1" on "msg.c" version "/main/1".
  ```

- Attach label **REL2** to version 3 on the **rel2_bugfix** branch of file **util.c**.

  cmd-context **mklabel –version /main/rel2_bugfix/3 REL2 util.c**

  ```
  Created label "REL2" on "util.c" version "/main/rel2_bugfix/3".
  ```

- Move label **REL2** to a different version of element **hello.c**, using a version-extended pathname to indicate that version.

  cmd-context **mklabel –replace REL2 hello.c@@/main/4**

  ```
  Moved label "REL2" on "hello.c" from version "/main/3" to "/main/4".
  ```

- Attach label **REL3** to each version that was used to build derived object **hello.o**. Note that both directories and files are labeled.

  cmd-context **mklabel –config hello.o REL3**

  ```
  Created label "REL3" on "/usr/hw/" version "/main/1".
  Created label "REL3" on "/usr/hw/src" version "/main/2".
  Created label "REL3" on "/usr/hw/src/hello.c" version "/main/3".
  Created label "REL3" on "/usr/hw/src/hello.h" version "/main/1".
  ```

- Attach label **REL5** to each C-language source file version that was used to build derived object **hello.**

  cmd-context **mklabel –config hello –name '*.c' REL5**

  ```
  Created label "REL5" on "/usr/hw/src/hello.c" version "/main/3".
  Created label "REL5" on "/usr/hw/src/util.c" version "/main/1".
  ```

- Attach label **REL5** to all versions in the VOB mounted at **/usr/hw** that were used to build derived object **hello**. Use interactive mode to enable use of the ClearCase and ClearCase LT "**...**" wildcard.

  cmd-context **mklabel –config hello –name ' /usr/hw/...' REL5**

```
Created label "REL5" on "/usr/hw/" version "/main/1".
Created label "REL5" on "/usr/hw/src" version "/main/2".
Created label "REL5" on "/usr/hw/src/hello.c" version "/main/3".
Created label "REL5" on "/usr/hw/src/hello.h" version "/main/1".
Created label "REL5" on "/usr/hw/src/util.c" version "/main/1".
```

SEE ALSO

mklbtype, **rmlabel**

mklbtype

Creates or updates a label type object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mklbtype [–rep·lace] [–glo·bal [–acq·uire] | –ord·inary] [–pbr·anch] [–sha·red]
[–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
label-type-selector ...

DESCRIPTION

The **mklbtype** command creates one or more *label types* with the specified names for future use within a VOB. After creating a label type in a VOB, you can attach labels of that type to versions of that VOB's elements, using **mklabel**.

Instance Constraints

The same version label can be attached to multiple versions of the same element. (The versions must all be on different branches. If two versions were labeled **JOHN_TMP** on branch **/main/bugfix**, the version-extended pathname **foo.c@@/main/bugfix/JOHN_TMP** would be ambiguous.) However, there are drawbacks to using the same version label several times in the same element:

- It is potentially confusing.
- In a version-extended pathname, you must always include a full branch pathname along with the version label (for example, **foo.c@@/main/rs6000_port/JOHN_TMP**).

By default, a new label type is constrained to use on only one version in an element's entire version tree. This allows you to omit the branch pathname portion of a version-extended pathname (for example, **foo.c@@/JOHN_TMP**). The **–pbranch** option relaxes this constraint, allowing the label type to be used once per branch.

Recommended Naming Convention

A VOB cannot contain a branch type and a label type with the same name. For this reason, we strongly recommend that you adopt this convention:

- Make all letters in names of branch types lowercase (**a** – **z**).
- Make all letters in names of label types uppercase (**A** – **Z**).

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: (with **–replace** only): type owner, VOB owner, , **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, label type (with **–replace** only).

OPTIONS AND ARGUMENTS

HANDLING OF NAME COLLISIONS. *Default:* An error occurs if a label type named *type-name* already exists in the VOB.

–rep·lace

Replaces the existing definition of *type-name* with a new one. If you do not include options from the existing definition, their values will be replaced with the defaults (Exception: the type's global scope does not change; you must explicitly specify **–global** or **–ordinary**).

If you specify a comment when using **–replace**, the comment appears in the event record for the modification (displayed with **lshistory –minor**); it does not replace the object's creation comment (displayed with **describe**). To change an object's creation comment, use **chevent**.

Constraints:

- You cannot replace either of the predefined label types **LATEST** and **CHECKEDOUT**.
- If there are existing labels of this type or if the containing VOB is replicated, you cannot replace a less constrained definition (**–pbranch** specified) with a more constrained definition. (The default is once per element.)
- When replacing a label type that was created with the **–shared** option, you must use **–shared** again; that is, you cannot convert a label type from shared to unshared.
- When converting a global type to ordinary, you must specify the global type as the *label-type-selector* argument. You cannot specify a local copy of the global type.

SPECIFYING THE SCOPE OF THE LABEL TYPE. *Default:* Creates an ordinary label type that can be used only in the current VOB.

–glo·bal [**–acq·uire**]

Creates a label type that can be used as a global resource by client VOBs in the

administrative VOB hierarchy. With **–acquire**, **mklbtype** checks all eclipsing types in client VOBs and converts them to local copies of the new global type.

For more information, see *Administering ClearCase*.

–ord·inary
Creates a label type that can be used only in the current VOB.

INSTANCE CONSTRAINTS. *Default:* A label of the new type can be attached to only one version of a given element.

–pbr·anch
Relaxes the default constraint, allowing the label type to be used once per branch in a given element's version tree. You cannot attach the same version label to multiple versions on the same branch.

MASTERSHIP OF THE LABEL TYPE. *Default:* Attempts to attach or remove labels of this type succeed only in the *VOB replica* that is the current *master* of the label type. The VOB replica in which the new label type is created becomes its initial master.

–sha·red
Allows you to create or delete labels of this type at any replica in the VOB family. If you also specify **–pbranch**, the replica must master the branch of the version you specify in the **mklabel** or **rmlabel** command. If you do not specify **–pbranch**, the replica must master the element of the version you specify in the **mklabel** or **rmlabel** command.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

NAMING THE LABEL TYPES. *Default:* The label type is created in the VOB that contains the current working directory unless you specify another VOB with the **@***vob-selector* argument.

label-type-selector ...
Names of the label types to be created. Specify *label-type-selector* in the form [**lbtype:**]*type-name*[**@***vob-selector*]

type-name	Name of the label type See the *Object Names* section in the **cleartool** reference page for rules about composing names.
vob-selector	VOB specifierf Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

See the section *Recommended Naming Convention on page 625*.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a label type that can be used only once per element. Provide a comment on the command line.

  cmd-context **mklbtype –c "Version label for V2.7.1 sources" V2.7.1**

  ```
  Created label type "V2.7.1".
  ```

- Create a label type that can be used once per branch in any element's version tree.

  cmd-context **mklbtype –nc –pbranch REL3**

  ```
  Created label type "REL3".
  ```

- Change the constraint on an existing label type so that it can be used once per branch. (This change does not affect existing labels of this type.)

  cmd-context **mklbtype –replace –pbranch –c "allow use on multiple branches" V2.7.1**

  ```
  Replaced definition of label type "V2.7.1".
  ```

SEE ALSO

describe, **lstype**, **mklabel**, **rename**, **rmtype**, **type_object**

mkpool

Creates a VOB storage pool or modifies its scrubbing parameters

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

- Create source pool:

  mkpool –sou·rce [–ln *pname* **]**
  [–c·omment *comment* **| –cfi·le** *comment-file-pname* **|–cq·uery | –cqe·ach | –nc·omment]**
  pool-selector ...

- Create derived object pool or cleartext pool:

  mkpool { **–der·ived | –cle·artext** } **[–ln** *pname* **]**
  [–siz·e *max-kbytes reclaim-kbytes* **[–age** *hours* **] [–ale·rt** *command* **]]**
  [–c·omment *comment* **| –cfi·le** *comment-file-pname* **|–cq·uery | –cqe·ach | –nc·omment]**
  pool-selector ...

- Update pool parameters:

  mkpool –upd·ate [–siz·e *max-kbytes reclaim-kbytes* **] [–age** *hours* **] [–ale·rt** *command* **]**
  [–c·omment *comment* **| –cfi·le** *comment-file-pname* **|–cq·uery | –cqe·ach | –nc·omment]**
  pool-selector ...

DESCRIPTION

The **mkpool** command creates a *source storage pool*, *derived object storage pool*, or *cleartext storage pool*, and initializes the pool's *scrubbing* parameters. You can also use this command to update the scrubbing parameters of an existing storage pool.

Storage pools are directories used as physical storage areas for different kinds of data:

- A *source storage pool* stores the *data containers* that contain versions of elements.
- A *derived object storage pool* stores *shared derived objects*—those that are referenced by more than one view.
- A *cleartext storage pool* is a cache of text files. If an element's versions are stored in a compressed format, accessing a particular version involves some processing overhead; a

type manager program is invoked to extract the *cleartext* of that version from the data container. As a performance optimization, the extracted version is cached as a file in a cleartext storage pool. The next access to that same version uses the cached copy, saving the cost of extracting the version from the data container again.

Creating a new VOB with the **mkvob** command creates one default pool of each kind: **sdft** (source pool), **ddft** (derived object pool), and **cdft** (cleartext pool).

By default, **mkpool** creates a storage pool as a directory within the VOB storage area. Source pools are always created within subdirectory **s** of the VOB storage directory; derived object pools are created within subdirectory **d**; cleartext pools are created within subdirectory **c**. The **–ln** option allows you to create pools elsewhere, to be accessed at the standard locations through symbolic links.

Pool Allocation and Inheritance

Each file element is assigned to one source pool and one cleartext pool. The source pool provides permanent storage, in one or more data container files, for all of the element's versions. If the element's versions are stored in a compressed format, the cleartext pool is used to cache extracted versions of that element, as described earlier. (If each version is stored uncompressed in a separate data container, the cleartext pool is not used.)

Each directory element is also assigned to one source pool and one cleartext pool. But directory versions themselves are not stored in these pools. (They are stored directly in the VOB database.) Rather, a directory's pool assignments are used solely for *pool inheritance*: each element created within the directory inherits its source and cleartext pool assignments.

Each directory element is also assigned to one derived object pool. All shared derived objects with pathnames in that directory are stored in that pool. A new directory element inherits the derived object pool of its parent, along with the source and cleartext pools.

The pool inheritance scheme begins at the VOB root directory (top-level directory element) created by **mkvob**, which is automatically assigned to the default pools.

You can change any of an element's pool assignments with the **chpool** command.

Scrubbing

Scrubbing is the process of reclaiming space in a derived object pool or cleartext pool. (Source pools are not subject to scrubbing.) This process is performed by the **scrubber** utility. **mkpool** initializes or updates these scrubbing parameters:

maximum size	(*max-kbytes*) Maximum pool size
reclaim size	(*reclaim-kbytes*) Size to which **scrubber** attempts to reduce the pool
age	(*hours*) Threshold to prevent premature scrubbing of recently referenced objects

The default settings for the scrubbing parameters are *max-kbytes* = 0, *reclaim-kbytes* = 0, *hours* = 96. See the **scrubber** reference page for details on how these parameters are interpreted.

By default, the scheduler runs **scrubber** periodically. See the **schedule** reference page for information on describing and changing scheduled jobs.

Getting Information on Storage Pools

The **lspool** command lists a VOB's storage pools. If you include the **–long** option, the current settings of the scrubbing parameters are listed, as well. (The **describe –pool** command displays the same information as **lspool –long**.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, *root* user. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, pool (for **-update** only).

OPTIONS AND ARGUMENTS

SPECIFYING THE KIND OF STORAGE POOL / SPECIFYING AN UPDATE. *Default:* You must specify the kind of pool, unless you use **–update** and name an existing pool. The following options are mutually exclusive.

–sou·rce
Creates a source pool.

–der·ived
Creates a derived object pool.

–cle·artext
Creates a cleartext pool.

–upd·ate
Asserts that the parameters of an existing pool are to be updated. You must also use a **–size** and/or **–age** option.

LOCAL VS. REMOTE STORAGE. *Default:* Creates a storage pool as a subdirectory under the VOB storage directory.

–ln *pname*
Creates a storage pool directory at *pname*, and creates *pool-name* in the VOB storage directory as a symbolic link to *pname*. You can create only one pool when using this option.

RESTRICTION: *pname* must be a full pathname, starting with a slash (/). It must also be a *global pathname*, valid on every host from which users will access the VOB. **mkpool** attempts to verify that this pathname is truly global, using a simple heuristic. (For

example, a pathname that begins with **/net** is likely to be global.) If it suspects that *pname* is not global, **mkpool** proceeds anyway, but displays a warning message:

```
Warning: Linktext for pool does not appear to be a global path.
```

This mechanism is independent of the *network storage registry* facility. Thus, the pathname to a remote storage pool directory must be truly global, not global within a particular network region.

CAUTION: We recommend that you keep source pools local, within the VOB storage directory. This strategy optimizes data integrity: a single disk partition contains all of the VOB's essential data. It also simplifies backup/restore procedures.

SPECIFYING NEW PARAMETERS. *Default:* For a new derived object or cleartext pool: the *maximum size* and *reclaim size* parameters are set to 0, which enables a special scrubbing procedure. (See the **scrubber** reference page.) The **age** parameter is set to 96 (hours). These parameters are meaningless for a source pool.

When updating an existing pool, you must use at least one of **–size** and **–age**.

–siz·e *max-kbytes reclaim-kbytes*
Specifies that the pool is scrubbed if its size exceeds *max-kbytes* KB; scrubbing will continues until the pool reaches the goal size of *reclaim-kbytes* KB.

–age *hours*
Prevents scrubbing of derived objects or cleartext files that have been referenced within the specified number of hours. Specifying **–age 0** restores the default age setting (96 hours).

SCRUBBER FAILURE PROCESSING. *Default:* If **scrubber** fails to scrub a pool below its *max-kbytes* level, it logs a warning message in **/var/adm/atria/log/scrubber_log**, but takes no other action.

–ale·rt *command*
Causes **scrubber** to run the specified command (typically, a shell script) whenever it fails to scrub a pool below its *max-kbytes* level.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE POOL. *Default:* Creates or updates a pool in the VOB containing the current working directory unless you specify another VOB with the *@vob-selector* suffix.

pool-selector ...

One or more names for the storage pools to be created. Specify *pool-selector* in the form [**pool:**]*pool-name*[**@***vob-selector*]

pool-name — Name of the storage pool

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a source pool that uses the default pool parameters.

  cmd-context **mkpool –source –c "pool for c source files" c_pool**

  ```
  Created pool "c_pool".
  ```

- Create a derived object pool with a maximum size of 10,000 KB (10 MB) and a reclaim size of 8,000 KB (8 MB). Allow the *age* parameter to assume its default value.

  cmd-context **mkpool –derived –nc –size 10000 8000 do1**

  ```
  Created pool "do1".
  ```

- Update the derived object pool created in the previous example, so that any derived object referenced within the last week (168 hours) is not scrubbed.

  cmd-context **mkpool –nc –update –age 168 do1**

  ```
  Updated pool "do1".
  ```

- Create a nonlocal cleartext storage pool at the globally accessible location **/usr/vobstore/ccase_pools/c2**, to be accessed as pool **cltxt2**.

  cmd-context **mkpool –nc –cleartext –ln /usr/vobstore/ccase_pools/c2 cltxt2**

  ```
  Created pool "cltxt2".
  ```

This command creates this symbolic link:

```
vob-storage-dir-pname/c/cltxt -> /usr/vobstore/ccase_pools/c2
```

- Create a cleartext pool named **my_ctpool** that uses the default pool parameters. Then, change all elements using pool **cdft** (the default cleartext pool) to use **my_ctpool** instead.

  cmd-context **mkpool -cleartext -c "alternate cleartext pool" my_ctpool**

  ```
  Created pool "my_ctpool".
  ```

  cmd-context **find . -all -element 'pool(cdft)' **
  -exec 'cleartool chpool -force my_ctpool $CLEARCASE_PN'

  ```
  Changed pool for "/usr/hw" to "my_ctpool".
  Changed pool for "/usr/hw/bin" to "my_ctpool".
  Changed pool for "/usr/hw/bin/hello" to "my_ctpool".
  Changed pool for "/usr/hw/bugs" to "my_ctpool".
  Changed pool for "/usr/hw/bugs/bug.report.21" to "my_ctpool".
  Changed pool for "/usr/hw/doc" to "my_ctpool".
  Changed pool for "/usr/hw/doc/util.doc" to "my_ctpool".
  Changed pool for "/usr/hw/include" to "my_ctpool".
  Changed pool for "/usr/hw/libs" to "my_ctpool".
  Changed pool for "/usr/hw/libs/libntx.a" to "my_ctpool".
  Changed pool for "/usr/hw/libs/libpvt.a" to "my_ctpool".
  .
  .
  .
  ```

SEE ALSO

chpool, **find**, **lspool**, **mkvob**, **schedule**, **scrubber**

mkproject

Create a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

mkproj·ect [**–c·omment** *comment* | **–cfi·le** *pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–tit·le** *title*] [**–mod·comp** *component-selector*[,...]]
–in *folder-selector*
[**–crm·enable** *ClearQuest-user-database-name*]
[*project-selector ...*]

DESCRIPTION

The **mkproject** command creates a UCM *project*. A project includes policy information and configuration information.

Projects are created in UCM folders. A folder or folder hierarchy should be in place before you create a project. If no folder exists, you can specify **RootFolder** as the folder selector with the **–in** option. **RootFolder** is a predefined object representing the parent folder of a UCM folder hierarchy. See **mkfolder** for more information.

Projects maintain a list of components that can be modified within the project. You can specify these with the **–modcomp** option. Streams in the project can make changes, such as checking out files, only in modifiable components; all other components are read-only.

See **chproject** for information on setting policy for a project.

Using Rational ClearQuest with UCM projects

Optionally, you can link a project to a Rational ClearQuest database. The schema of the ClearQuest database must be UCM-enabled, and your system must be configured for the correct schema repository. All ClearQuest-enabled projects in the same project VOB must link to the same ClearQuest user database.

See **chproject** for related information.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required.

Locks: An error occurs if any of the following objects are locked: UCM project VOB.

Mastership: There are no mastership requirements.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–c**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING A TITLE FOR THE PROJECT. *Default*: The project's name, as specified by the *project-selector* argument.

–tit·le *title*
Specifies a project title applied to all projects created with this command. The *title* argument can be a character string of any length. Use double quotes to enclose a multiple-word title or a title with special characters.

SPECIFYING A FOLDER FOR THE PROJECT. *Default*: None.

–in *folder-selector*
Specifies a folder.

folder-selector is of the form: [**folder:**]*folder-name*[*@vob-selector*] and *vob* is the folder's UCM project VOB.

SPECIFYING MODIFIABLE COMPONENTS. *Default*: None.

–mod·comp *component-selector*[,...]
Specifies the components that can be modified by this project.

SPECIFYING A LINK TO THE CLEARQUEST DATABASE. *Default*: None.

–**crm·enable** *ClearQuest-user-database-nam*e
Enables a link from the project to the specified Rational ClearQuest database. The schema of the ClearQuest database must be UCM-enabled and your system must be configured for the correct schema repository.

SPECIFYING THE PROJECT NAME. *Default*: A generated name.

project-selector
Specifies the project.

project-selector is of the form: [**project:**]*project-name*[*@vob-selector*] and *vob* is the project's UCM project VOB.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a project in the **RootFolder** of the project VOB **webo_pvob**.

cmd-context **mkproject -c "creating webo project release 1" **
-title webo_proj1 -in webo_projects@/vobs/webo_pvob webo_proj1@/vobs/webo_pvob

```
Created project "webo_proj1".
```

SEE ALSO

chproject, **lsproject**, **mkfolder**, **rmproject**

mkregion

Registers a new ClearCase network region

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

mkregion –tag *region-tag* [**–tco·mment** *tag-comment*] [**–rep·lace**]

DESCRIPTION

The **mkregion** command registers a new network region by adding a new *region-tag* (region name) and, optionally, a comment to the file **/var/adm/atria/rgy/regions** on the ClearCase registry server host. Use the **lsregion** command to display the *region-tag*s contained in **regions**.

After creating a new region, you can create VOB-tags and view-tags for the region with **mktag**, **mkvob**, and **mkview**.

A ClearCase client host (which may also be an Attache helper host) can belong to only one region. Use the **hostinfo –long** command to display the client host's registry region. See **registry_ccase** and *Administering ClearCase* for more information on ClearCase network regions.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE REGION TAG. *Default:* None. You must name the region.

–tag *region-tag*
Names the region. *region-tag* can be up to 32 characters.

–tco·mment *tag-comment*
Adds a comment to the *region-tag*'s entry in the registry file. Use **lsregion –long** to display the *tag-comment*.

OVERWRITING AN EXISTING TAG. *Default:* An error occurs if **mkregion** names a *region-tag* that already exists.

–rep·lace
Replaces the *tag-comment* of an existing *region-tag*. No error occurs if the *region-tag* does not exist. You cannot use **–replace** to change an existing *region-tag;* to do so, you must

first delete the existing tag with **rmregion –tag**, and then create a new one with **mkregion –tag**.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Register a new region with tag **us_east**.

  cmd-context **mkregion –tag us_east –tcomment "all east coast ClearCase hosts"**

- Change the comment stored with region-tag **us_east**.

  cmd-context **mkregion –tag us_east –tcomment "east coast development hosts" –replace**

FILES

/var/adm/atria/rgy/regions
/var/adm/atria/rgy/rgy_region.conf

SEE ALSO

lsregion, **lsview**, **mktag**, **mkview**, **mkvob**, **registry_ccase**, **rmregion**

mkstgloc

Creates a server storage location for views or VOBs.

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

- ClearCase only:

  mkstgloc { **–vie·w** | **–vob** } [**–f·orce**] [**–c·omment** *comment*]
  [**–reg·ion** *network-region*]
  [**–hos·t** *hostname* **–hpa·th** *host-storage-pname* **–gpa·th** *global-storage-pname*
  | **–ngp·ath** [**–hos·t** *hostname* **–hpa·th** *host-storage-pname*]]
  stgloc-name stgloc-pname

- ClearCase LT only:

  mkstgloc { **–vie·w** | **-vob** } [**–f·orce**] [**–c·omment** *comment*]
  stgloc-name stgloc-pname

DESCRIPTION

The **mkstgloc** command creates and registers a named server storage location for view- or VOB-storage directories. By "creates and registers" we mean the command initializes a physical directory and writes information describing that directory to the ClearCase or ClearCase LT registry. For information on the registry, see the **registry_ccase** reference page.

Other Uses for mkstgloc

In addition to creating new server storage locations, you can use **mkstgloc** to

- Adopt an existing directory as a server storage location—an existing directory is adopted if *stgloc-pname* specifies that directory.
- (ClearCase only) Register an existing server storage location in a new region—a server storage location is registered in a new region if *stgloc-pname* specifies an existing server storage location. Specify new arguments for options such as **–region** and **–host** as appropriate for the region in which you are registering the server storage location.

Default Selection of Server Storage Locations During View and VOB Creation

Refer to the **mkview** and **mkvob** reference pages for information on the default selection of server storage locations in view and VOB creation.

ClearCase Only—File System Connectivity Considerations

Before creating a server storage location for a ClearCase view or VOB, determine whether there is file system connectivity between the server storage location's host and its clients in the regions that advertise the server storage location. File system connectivity determines how you can use the server storage location, as follows:

Server Storage Location Use	File System Connectivity
Dynamic views	Required (a global path to the server storage location must exist)
VOB to be accessed through dynamic views	Required (a global path to the server storage location must exist)
Snapshot views	Not required
VOB to be accessed only through snapshot views	Not required

ClearCase Only—Derived and Explicitly Specified Client Accessibility Information

To be accessible to its clients, a ClearCase server storage location needs to be registered with the following information:

- The name of the host where the server storage location resides.
- A host-local pathname to the server storage location.
- For dynamic views or VOBs accessed through dynamic views, a global pathname to the server storage location relative to the host's network region.
- The network region in which the host resides.

In many cases, ClearCase heuristically derives appropriate accessibility information from the *stgloc-pname* argument. In cases where there is no file-system connectivity between the server storage location and its clients, ClearCase derives the host name and host-local path, but because no meaningful global path can be derived, you must specify **–ngpath** to unset the global path information.

An unusual network configuration may defeat the heuristic by which accessibility information is derived, thereby preventing access to the server storage location by some or all ClearCase clients. In such cases, set the registry information explicitly, following these guidelines:

- To create a server storage location for dynamic views or for VOBs intended to be accessed through dynamic views, use the option set, –**host –hpath –gpath**.
- To create a server storage location for snapshot views or for VOBs intended to be accessed only through snapshot views, use:
  - **–host –hpath –gpath** when there is file-system connectivity between the server storage location host and its clients.
  - **–ngpath –host –hpath** when there is no file-system connectivity between the server storage location host and its clients.

ClearCase LT Only—File System Connectivity and Client Accessibility

For ClearCase LT, issues related to file system connectivity and client accessibility to server storage locations are not as complex as they can be for ClearCase. ClearCase LT assumes there is no filesystem connectivity such as that provided by NFS, so there are no command options or arguments related to the presence or absence of file system connectivity.

All server storage locations reside at the ClearCase LT server host and its clients learn the name of that host at client-install time. In rare cases, the host chosen to serve as the ClearCase LT server host is a "multihome" host—a host that is known by different names through different network interfaces. However, ClearCase LT requires that the ClearCase LT server host be known to all its clients by the same host name. Therefore, you must set up the host's network configuration to ensure that a single host name maps to different network addresses that are appropriate for the various client hosts of the server. See *Administering ClearCase* for more information.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE OBJECT TYPE FOR WHICH A SERVER STORAGE LOCATION IS TOBE CREATED. *Default:* None.

–vie·w
Specifies that the server storage location is for view storage directories.

–vob
Specifies that the server storage location is for VOB storage directories.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the server storage location is to be created as specified only if you are adopting an existing directory (see *Other Uses for mkstgloc*).

–f·orce
Suppresses the confirmation step.

COMMENTS. *Default:* None.

–c·omment *comment*
Specifies a comment for the server storage location's entry in the registry. Use **lsstgloc** to display the comment.

SPECIFYING A NETWORK REGION. *Default:* The host's network region.

–reg·ion *network-region*
Causes the server storage location to be registered in the specified *network region*. An error occurs if the region does not exist.

SPECIFYING NETWORK ACCESSIBILITY. *Default:* A host name, host-local path, and global path are derived from the specified *stgloc-pname*.

–hos·t *hostname*
–hpa·th *host-storage-pname*
–gpa·th *global-storage-pname*
–ngp·ath
Use these options only after you have determined that you need to explicitly set a server storage location's registry information (see *ClearCase Only—Derived and Explicitly Specified Client Accessibility Information*). The information is written to the registry exactly as you specify it.

You must either specify the **–host**, **–hpath**, and **–gpath** options as a set; or use **–ngpath** and optionally specify **–host** and **–hpath**.

–host *hostname*—The name of the host where the server storage location is to reside.

–hpath *host-storage-pname*—A standard full pathname to the server storage location that is valid on the specified host.

–gpath *global-storage-pname*—A standard full pathname to the server storage location that is valid in the target network region for all client hosts that are to access the server storage location.

–ngp·ath—Specifies that in the target region there is no global path by which the server storage location can be accessed.

SPECIFYING A NAME AND PATH FOR THE SERVER STORAGE LOCATION. *Default:* None.

stgloc-name
Specifies the name under which the server storage location is to be registered. The name must be unique within the target region.

stgloc-pname

Specifies the path to the server storage location.

ClearCase only—*stgloc-pname* must specify a location on a host where there is an installation of ClearCase that is not a client-only installation. For storage intended for snapshot views or VOBs to be accessed only through snapshot views, *stgloc-pname* must be a UNC name if and only if there is a global path to the server storage location (that is, you did not specify **–ngpath**).

ClearCase LT only—*stgloc-pname* must be located on the ClearCase LT server host and must be must be a UNC name.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Create a server storage location for VOBs that dynamic views can access, allowing **mkstgloc** to derive client accessibility information from the specified server storage location pathname.

  cmd-context **mkstgloc –vob stgloc_vob1 ~/stgloc_vob1**
  ```
  Created and advertised Server Storage Location.
  Host-local path: peroxide:/export/home/bert/stgloc_vob1
  Global path: /net/peroxide/export/home/bert/stgloc_vob1
  ```

- Create a server storage location for dynamic views, allowing **mkstgloc** to derive client accessibility information from the specified server storage location pathname.

  cmd-context **mkstgloc –view stgloc_view1 ~/stgloc_view1**
  ```
  Created and advertised Server Storage Location.
  Host-local path: peroxide:/export/home/bert/stgloc_view1
  Global path: /net/peroxide/export/home/bert/stgloc_view1
  ```

- Create a server storage location for a VOB that only snapshot views will access.

  cmd-context **mkstgloc –vob –ngpath store1 ~/store1**
  ```
  Created and advertised Server Storage Location.
  Host-local path: peroxide:/export/home/bert/store1
  Global path: <no-gpath>
  ```

SEE ALSO

lsstgloc, **mkview**, **mkvob**, **registry_ccase**, **rmstgloc**

mkstream

Creates a stream for a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

mkstream [–c·omment *comment* | **–cfi·le** *pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–tit·le** *title*] [**–int·egration**]
[**–bas·eline** *baseline-selector*[,...]]
–in *project-selector* [*stream-selector*...]

DESCRIPTION

The **mkstream** command creates a stream for use with a UCM project. A stream consists of a title, a set of baselines that configure the stream, and a record of the set of activities associated with the stream.

There are two kinds of streams with UCM projects:

- As a shared work area for integrating work from different sources. This is called the project's *integration stream*. Each project has exactly one integration stream.
- As an isolated work area for use in active code development. This is called a *development stream*. A project can have any number of development streams.

To create a stream, you must specify its project and whether it is an integration stream or development stream. Note that a project's integration stream must be present before a development stream can be created.

Optionally, you can assign the stream a title and a set of *foundation baselines*. Foundation baselines specify a stream's configuration by selecting the file and directory versions that are accessible in the stream.

Streams are accessed through views (see **mkview –stream**). Typically, a project's integration stream has a view for each developer, whereas each development stream has a single view.

A stream can have more than one view attached to it. In general, because project members work with a common integration stream, the stream has several views attached to it. A development stream usually has only one view attached to it.

PERMISSIONS AND LOCKS

Permissions Checking: None.

Locks: An error occurs if there are locks on any of the following objects: the UCM project VOB, the project.

Mastership: There are no mastership requirements.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–c**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE STREAM TITLE. Default: A generated title.

–tit·le *title*
Assigns the specified title to all streams created.

STREAM CONFIGURATION. *Default:* The stream's configuration is empty (that is, it has no foundation baselines).

–baseline *baseline-selector*[,...]
Specifies one or more baselines to use as the stream's initial configuration—you can subsequently use **rebase** to change the stream's configuration.

baseline-selector is of the form: [**baseline:**]*baseline-name*[*@vob-selector*] and *vob* is the baseline's UCM project VOB.

The following restrictions apply to the specified baselines:

- For a development stream, all foundation baseline must either be baselines created in the project's integration stream, or serve as the integration stream's foundation baselines.
- For an integration stream, all foundation baselines must be either baselines created in other projects' integration streams, or be import or initial baselines. You cannot use baselines created in development streams.

SPECIFYING THE STREAM'S ROLE IN THE PROJECT. *Default:* Development stream.

–int·egration
Creates an integration stream, which is used for shared elements on a project and as a source for recording baselines. Each project can have one integration stream.

SPECIFYING THE STREAM'S PROJECT. *Default:* None.

–in *project-selector*
Specifies the stream's project.

project-selector is of the form: [**project:**]*project-name*[*@vob-selector*] and *vob* is the project's UCM project VOB.

SPECIFYING THE STREAM NAME. *Default:* A generated name.

stream-selector ...
Specifies a stream name.

You can specify the stream as a simple name or as an object selector of the form [**stream**]**:***name***@***vob-selector*, where *vob-selector* specifies a project VOB (see the **cleartool** reference page). If you specify a simple name and the current directory is not a project VOB, then this command assumes the stream resides in the project VOB associated with the current view. If the current directory is a project VOB, then that project VOB is the context for identifying the stream.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a development stream for the **webo** project.

```
cmd-context mkstream -title chris_webo_dev \
-in webo_proj1@/vobs/webo_pvob chris_webo_dev@/vobs/webo_pvob

Created stream "chris_webo_dev".
```

- Create an integration stream.

```
cmd-context mkstream -title integration -integration \
-in webo_proj1 integration@/vobs/webo_pvob

Created stream "integration".
```

- Join a project. This example shows the sequence of commands to follow to join a UCM project.

a. Find the *project-selector* for the project you want to join. For example:

cmd-context **lsproject –invob /vobs/webo_pvob**

```
01-Mar-00.16:31:33  webo_proj1  ktessier   "webo_proj1"
05-Jun-00.12:31:33  webo_proj2 ktessier   "webo_proj2"
```

b. Create your development stream. For example:

cmd-context **mkstream –title chris_webo_dev **
**-in webo_proj1@/vobs/webo_pvob –baseline BL3@/vobs/webo_pvob **
chris_webo_dev@/vobs/webo_pvob

```
Created stream "chris_webo_dev".
```

c. Create a view attached to your development stream:

cmd-context **mkview -stream chris_webo_dev@/vobs/webo_pvob **
-tag chris_webo_dev /export/views/chris_webo_dev.vws

```
Created view.
Host-local path: venus:/export/views/chris_webo_dev.vws
Global path: /net/venus/export/views/chris_webo_dev.vws
It has the following rights:
User : chris : rwx
Group: user     : rwx
Other:          : r-x
Attached view to stream "chris_webo_dev".
```

d. Create a view attached to the project's integration stream:

cmd-context **mkview -stream integration@/vobs/webo_pvob **
-tag webo_integ /export/views/webo_integ.vws

SEE ALSO

chstream, **lsstream**, **rebase**, **rmstream**

mktag

Creates a tag for a view or VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only—Create a tag for a dynamic view:

  mktag –vie·w –tag *dynamic-view-tag* [**–tco·mment** *tag-comment*]
  [**–rep·lace** | **–reg·ion** *network-region*] [**–nst·art**] [**–nca·exported**]
  [**–hos·t** *hostname* **–gpa·th** *global-storage-pname*] *dynamic-view-storage-pname*

- ClearCase and Attache only—Create a tag for a snapshot view:

  mktag –vie·w –tag *snapshot-view-tag* [**–tco·mment** *tag-comment*]
  [**–rep·lace** | **–reg·ion** *network-region*] [**–nst·art**]
  [**–hos·t** *hostname* **–gpa·th** *global-storage-pname*
  | **–ngpath** [**–host** *hostname*]] *snapshot-view-storage-pname*

- ClearCase and Attache only—Create a VOB-tag:

  mktag –vob –tag *vob-tag* [**–tco·mment** *tag-comment*]
  [**–rep·lace** | **–reg·ion** *network-region*] [**–opt·ions** *mount-options*]
  [**–pub·lic**] [**–pas·sword** *tag-registry-password*] [**–nca·exported**]
  [**–hos·t** *hostname* **–gpa·th** *global-storage-pname*
  | **–ngp·ath** [**–hos·t** *hostname*]] *vob-storage-pname*

- ClearCase LT only—Create a view-tag:

  mktag –vie·w –tag *view-tag* [**–tco·mment** *tag-comment*] [**–rep·lace**] [**–nst·art**]
  snapshot-view-storage-pname

- ClearCase LT only—Create a VOB-tag:

  mktag –vob –tag *vob-tag* [**–tco·mment** *tag-comment*] [**–rep·lace**] *vob-storage-pname*

mktag

DESCRIPTION

For an existing *view* or *VOB*, the **mktag** command creates or replaces an entry in the registry. A view or VOB gets one tag when it is created with **mkview** or **mkvob**.

ClearCase and Attache Only—Using mktag

In ClearCase and Attache, you can use **mktag** to create additional tags, enabling access from multiple *network regions*. Each network region needs its own tag for a view or VOB. A single region cannot have multiple tags for the same VOB. (Multiple tags for a view are valid, but not recommended.) However, a single tag can be assigned to multiple regions with multiple **mktag** commands. See the **registry_ccase** reference page for a discussion of network regions.

By default, creating a view-tag activates the view on your host, by implicitly performing a **startview** command. This does not occur if your host is not in the tag's assigned network region, or if you use the **–nstart** option. For a *dynamic view*, creating the view-tag also activates the view. However, creating a VOB-tag does not activate the VOB; use **mount** for this purpose.

ClearCase LT Only—Using mktag

In ClearCase LT, **mktag** is used to replace a tag. **mktag –view** activates the view unless you use the **–nstart** option.

PERMISSIONS AND LOCKS

Permissions checking: In ClearCase and Attache, you must be the VOB owner to create a private VOB-tag. In ClearCase LT, no special permissions are required.

Locks: None apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE KIND OF TAG TO REPLACE. *Default:* None.

–vie·w
Specifies a view-tag.

–vob
Specifies a VOB-tag.

SPECIFYING THE TAG. *Default:* None.

–tag *dynamic-view-tag* | *snapshot-view-tag*
A name for the view, in the form of simple filename.

–tag *vob-tag*
ClearCase and Attache only—Either a standard full pathname, which specifies the location at which the VOB will be mounted; or a name for the VOB, in the form of an absolute single-component pathname (for example, **/big_vob**). If the region is a

ClearCase LT region (these regions are named **CCLT**), then the VOB tag must be in the form of an absolute single-component pathname.

ClearCase LT—a name for the VOB, in the form of an absolute single-component pathname; for example, **/big_vob**.

–tco·mment *tag-comment*
Adds a comment to the tag's entry in the registry. Use the **–long** option with **lsvob** or **lsview** to display the tag comment.

OVERWRITING AN EXISTING TAG. *Default:* None.

–rep·lace
Replaces an existing tag registry entry with a new entry. (No error occurs if the tag does not exist.) You can use this option to change the tag comment and access paths. You cannot use **–replace** to change an existing tag's name; to do this, delete the tag with **rmtag** and then use **mktag**.

ClearCase and Attache only—This option also enables you to convert private VOBs to public and vice versa, and to change **startview** behavior. (To change a private VOB to public, you must provide the tag-registry password. To change a public VOB to private, you must be the VOB owner.)

STARTING THE VIEW. *Default for ClearCase and Attache:* Starts the **view_server** process on the host where the view storage location resides, if it isn't already running. For a dynamic view, creating a view tag also makes the view active on your host, making the view tag appear as a directory entry in the viewroot directory, **/view**. *Default for ClearCase LT:* Starts the **view_server** process on the ClearCase LT server host.

–nst·art
Suppresses starting of the **view_server** process.

MARKING A VIEW OR VOB FOR EXPORT. *Default:* The view-tag or VOB-tag is not marked for export.

–nca·exported
Marks the view-tag or VOB-tag in the registry as an export view or VOB. See the **mkview** and **mkvob** reference pages for more information. This option applies to dynamic view environments only.

SPECIFYING A NETWORK REGION. *Default:* Creates a tag in the local host's network region. (Use the **hostinfo –long** command to list a host's network region.) See the **registry_ccase** reference page for a discussion of *network regions*.

–reg·ion *network-region*
Creates the tag in the specified *network region*. An error occurs if the region does not already exist. An error occurs if the VOB already has a tag in the specified network region.

SPECIFYING MOUNT OPTIONS. *Default:* No mount options are included in the VOB registry entry for a new VOB-tag.

–opt·ions *mount-options*
(VOB-tags only. You must be **root** to use this option.) Specifies mount options to be invoked when the VOB is activated through this VOB-tag. See the **mkvob** reference page for syntax details.

PUBLIC VS. PRIVATE VOB. *Default:* Creates a private VOB-tag (does not apply to view-tags). An error occurs if you are not the VOB owner.

–pub·lic
Creates a public VOB-tag. See the **mkvob** reference page for a discussion of public and private VOBs.

–pas·sword *tag-registry-password*
Specifies the *VOB-tag password*, which is required to create a public tag or to create a private tag when you include **suid** as an argument to **–options**.

In these cases, if you do not include a password, you are prompted for it. The value you specify is checked against the tag registry password; an error occurs if there is no match. For more information, see the **registry_ccase** reference page.

NOTE: The VOB-tags for a given VOB must all be *private*, or all be *public*.

SPECIFYING CLIENT ACCESSIBILITY INFORMATION. *Default:* Derived from *dynamic-view-storage-pname* or *snapshot-view-storage-pname* for a view tag, or from *vob-storage-pname* for a VOB tag.

–hos·t *hostname*
–gpa·th *global-pname*
–ngp·ath
See the **mkstgloc** reference page for general information on these options; note, however, that the view or VOB for which you are making a tag need not necessarily reside in a server storage location created with **mkstgloc**.

The information you provide is written to the registry exactly as you specify it.

SPECIFYING THE PATH TO THE VOB OR VIEW STORAGE. *Default:* None.

dynamic-view-storage-pname
snapshot-view-storage-pname
vob-storage-pname
Specifies the path to an existing storage directory for a view or a VOB (the directory may be in a server storage location; see **mkstgloc**).

ClearCase and Attache only—The pathname must specify a location on a host where there is an installation of ClearCase that is not a client-only installation. For storage

intended for snapshot views or VOBs to be accessed only through snapshot views, the pathname must be a UNC name if and only if there is a global path to the server storage location (that is, you have not specified **–ngpath**).

ClearCase LT only—The pathname must be located on the ClearCase LT server host and must be must be a UNC name.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

For the network region **europe**, assign the new view-tag **view5** to an existing view storage area.

cmd-context **mktag -view -tag view5 -region europe /net/gw/host3/view_store/view5.vws**

- For the network region **europe**, register an existing VOB with a public VOB-tag.

  cmd-context **mktag -vob -tag /vobs/us_east1 -region europe -public **
  -password tagPword /net/gw/host2/vob_store/vob1.vbs

- Convert a private VOB to a public VOB, by replacing its private VOB-tag with a public one.

  cmd-context **mktag -vob -tag /vobs/publicvob -replace -public **
  -pass tagPword /vobs/private.vbs

- Mark an existing view and VOB for export.

  cmd-context **mktag -view -tag bugfix -replace -ncaexported /net/neon/views/bugfix.vws**

  cmd-context **mktag -vob -tag /vobs/dev -replace -ncaexported /net/pluto/vobstore/dev.vbs**

SEE ALSO

lsview, **lsvob**, **mkstgloc**, **mkview**, **mkvob**, **registry_ccase**, **rmtag**, **startview**, **view_server**, **vob_server**

mktrigger

Attaches a trigger to an element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

mktrigger [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery**
| **–cqe·ach** | **–nc·omment**]
[–r·ecurse] [**–nin·herit** | **–nat·tach**] [**–f·orce**]
trigger-type-selector pname ...

DESCRIPTION

Prerequisite: A *trigger type* object, created with **mktrtype –element**, must already exist in the VOBs containing the specified elements.

The **mktrigger** command attaches a *trigger* to one or more elements. An attached trigger *fires* (executes the trigger action) when the element or any of its versions is involved in an operation specified in the trigger type definition. For example, if a trigger type is defined to fire on a **checkin** command, the attached trigger fires when the specified element is checked in. If a VOB operation causes multiple attached triggers to fire, the order of firing is undefined.

Trigger Inheritance

By means of a *trigger inheritance* scheme, newly created elements (but not existing elements) inherit the triggers that are currently associated with their parent directory element. But a simple inherit-all-triggers strategy does not suit the needs of many sites. For example:

- You may want some of a directory's triggers not to propagate to its subtree.
- You may want some triggers to fire only for file elements, not for directory elements.

To enable such flexibility, each directory element has two independent lists of trigger types:

- Its *attached list* specifies triggers that fire on operations involving the directory element.
- Its *inheritance list* specifies triggers that elements created within the directory inherit.

By default, attaching a trigger to a directory element updates both lists:

cmd-context **mktrigger trig_co proj**

```
Added trigger "trig_co" to inheritance list of "proj".
Added trigger "trig_co" to attached list of "proj".
```

Each file element has only an attached list:

cmd-context **mktrigger trig_co util.c**

```
Added trigger "trig_co" to attached list of "util.c".
```

You can use the **–ninherit** and **–nattach** options to control exactly which triggers on a directory element are inherited. (And you can make adjustments using the **–ninherit** and **–nattach** options of the **rmtrigger** command.)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, trigger type.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

ATTACHING TRIGGERS TO AN ENTIRE SUBDIRECTORY TREE. *Default:* If a *pname* argument names a directory element, the trigger is attached only to the element itself, not to any of the existing elements within it.

–r·ecurse
Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are not traversed during the recursive descent into the subtree.

CONTROLLING TRIGGER INHERITANCE. *Default:* For a directory element, the specified trigger type is placed both on the element's attached list and its inheritance list. (For a file element, the trigger type is placed on its attached list, which is its only trigger-related list.) The following options apply to directory elements only.

–nin·herit
The trigger is placed on the element's attached list, but not on its inheritance list. This option is useful when you want to monitor operations on a directory, but not operations on the files within the directory.

–nat·tach
The trigger is placed on the element's inheritance list, but not on its attached list. This option is useful when you want to monitor operations on the files within a directory, but not operations on the directory itself.

OBSERVING ELEMENT TYPE RESTRICTIONS. *Default:* If *trigger-type-name* is defined with a restriction to one or more element types, **mktrigger** refuses to process an element of another type.

–f·orce
Attaches a trigger to an element whose type does not match the definition of the trigger type. Such a trigger does not fire unless you change the element's type (**chtype**) or you redefine the trigger type (**mktrtype –replace**).

SPECIFYING THE TRIGGER TYPE. *Default:* None.

trigger-type-selector
The name of an existing *element* trigger type. Specify *trigger-type-selector* in the form [**trtype:**]*type-name*[**@***vob-selector*]

type-name — Name of the trigger type

vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

SPECIFYING THE ELEMENTS. *Default:* None.

pname ...
One or more pathnames, specifying elements to which the specified trigger type is to be attached.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Attach a trigger to element **hello.c**.

  cmd-context **mktrigger trig1 hello.c**
  ```
  Added trigger "trig1" to attached list of "hello.c".
  ```

- Attach a trigger to element **util.c**, even if its element type does not appear in the trigger type's restriction list.

  cmd-context **mktrigger -force trig1 util.c**
  ```
  Added trigger "trig1" to attached list of "util.c".
  ```

- Attach a trigger to directory element **src**.

  cmd-context **mktrigger trig1 src**
  ```
  Added trigger "trig1" to attached list of "src".
  Added trigger "trig1" to inheritance list of "src".
  ```

- Add a trigger to the **release** directory's inheritance list, but not to its attached list.

  cmd-context **mktrigger -nattach trig1 release**
  ```
  Added trigger "trig1" to inheritance list of "release".
  ```

SEE ALSO

describe, **mktrtype**, **rmtrigger**

mktrtype

Creates a trigger type object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- Create element trigger type:

  mktrtype –ele·ment [**–a·ll**] [**–rep·lace**]
  { **–pre·op** | **–pos·top** } *opkind*[,...] [**–nus·ers** *login-name*[,...]]
  { **–exe·c** *command*
  | **–execu·nix** *command*
  | **–execw·in** *command*
  | **–mkl·abel** *label-type-selector*
  | **–mka·ttr** *attribute-type-selector*=*value*
  | **–mkh·link** *hlink-type-selector*,**to=***pname*
  | **–mkh·link** *hlink-type-selector*,**from=***pname* } ...
  [*restriction-list*]
  [**–pri·nt**]
  [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
  type-selector ...

- Create type trigger type:

  mktrtype –typ·e [**–rep·lace**] { **–pre·op** | **–pos·top** } *opkind*[,...]
  [**–nus·ers** *login-name*[,...]]
  { **–exe·c** *command*
  | **–execu·nix** *command*
  | **–execw·in** *command*
  | **–mkl·abel** *label-type-selector*
  | **–mka·ttr** *attribute-type-selector*=*value*
  | **–mkh·link** *hlink-type-selector*,**to=***pname*
  | **–mkh·link** *hlink-type-selector*,**from=***pname* } ...
  inclusion-list [**–pri·nt**]

[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**] *type-selector ...*

- A *restriction-list* contains one or more of:

  –att·ype *attr-type-selector*[,...] **–hlt·ype** *hlink-type-selector*[,...]
  –brt·ype *branch-type-selector*[,...] **–lbt·ype** *label-type-selector*[,...]
  –elt·ype *elem-type-selector*[,...] **–trt·ype** *trigger-type-selector*[,...]

  NOTE: **–xxtype aaa,bbb** is equivalent to **–xxtype aaa –xxtype bbb**.

- An *inclusion-list* contains one or more of:

  –att·ype *attr-type-selector*[,...] or **–att·ype –all**
  –brt·ype *branch-type-selector*[,...] or **–brt·ype –all**
  –elt·ype *elem-type-selector*[,...] or **–elt·ype –all**
  –hlt·ype *hlink-type-selector*[,...] or **–hlt·ype –all**
  –lbt·ype *label-type-selector*[,...] or **–lbt·ype –all**
  –trt·ype *trigger-type-selector*[,...] or **–trt·ype –all**

  NOTE: **–xxtype aaa,bbb** is equivalent to **–xxtype aaa –xxtype bbb**.

DESCRIPTION

The **mktrtype** command creates one or more *trigger types* for use within a VOB. A trigger type defines a sequence of one or more trigger actions to be performed when a specified ClearCase, ClearCase LT, or Attache operation occurs. The set of operations that initiates each trigger action—causes the trigger to fire—can be very limited (for example, **checkout** only) or quite general (for example, any operation that modifies an element). You can use a *restriction list* to further limit the circumstances under which a trigger action is performed.

Only a VOB's owner or the *root* user can create a trigger type.

There are three kinds of trigger types:

- An *element trigger type* works like a label type or attribute type: an instance of the type (that is, a *trigger*) must be explicitly attached to one or more individual elements with the **mktrigger** command. The trigger actions are performed when the specified operation is invoked on any of those elements. An element must exist before the trigger can be attached. (This means that putting a trigger on a **mkelem** operation has no effect.)
- A variant of the above, called an *all-element trigger type*, is associated with the entire VOB. (Hence, no **mktrigger** command is required.) In effect, an instance of the type is implicitly attached to each element in the VOB, even those created after this command is executed. This trigger type is useful for disallowing creation of elements that have certain characteristics.

- A *type trigger type* is associated with one or more type objects. The trigger actions are performed when any of those type objects is created or modified.

Unlike other types, trigger types cannot be global.

Trigger Firing

Causing a set of trigger actions to be performed is termed *firing a trigger*. Each trigger action can be either of the following:

- Any command (or sequence of commands) that can be invoked from a shell. A command can use special environment variables (EVs), described in the *Trigger Environment Variables* section, to retrieve information about the operation.
- Any of several built-in actions defined by **mktrtype**. The built-in actions attach metadata annotations to the object involved in the operation.

Trigger actions execute with the user-ID of the process that caused the trigger to fire.

Interactive Trigger Action Scripts. A command shell script executed as (part of) a trigger action can interact with the user. The **clearprompt** utility is designed for use in such scripts; it can handle several kinds of CLI-style and GUI-style user interactions.

Multiple Trigger Firings. A single operation can cause any number of triggers to fire. The firing order of such simultaneous triggers is indeterminate. If multiple trigger operations must be executed in a particular order, use a single trigger defining all of the operations their order of execution.

It is also possible for triggers to create a chain reaction. For example, a checkin operation fires a trigger that attaches an attribute to the checked-in version; the attach attribute operation, in turn, fires a trigger that sends mail to an administrator. You can use the **CLEARCASE_PPID** environment variable to help synchronize multiple firings (for more information, see *Trigger Environment Variables* on page 670).

If a trigger is defined to fire on a hyperlink operation, and the hyperlink connects two elements, the trigger fires twice—once for each end of the hyperlink.

Suppressing Trigger Firing. The firing of a trigger can be suppressed when the associated operation is performed by certain users. Firing of an all-element trigger is suppressed if the trigger type has been made obsolete. (See the **lock** reference page).

Trigger Interoperation

The **–execunix** and **–execwin** options allow a single trigger type to have different paths for the same script, or completely different scripts, on UNIX and Windows hosts. When the trigger is fired on UNIX, the command specified with **–execunix** runs; when the trigger is fired on Windows, the command specified with **–execwin** runs.

Triggers with only **–execunix** commands always fail on Windows. Likewise, triggers that only have **–execwin** commands fail when they fire on UNIX.

The **–exec** option, whose command will run on both platforms, can be used in combination with the platform-specific options. For example, you can cascade options:

–exec *arg1* **–execunix** *arg2* **–execwin** *arg3* **–mklabel** *arg4* ...

PREOPERATION AND POSTOPERATION TRIGGERS

A preoperation trigger (**–preop** option) fires before the corresponding operation begins. The one or more actions you've specified take place in their order on the command line.

This type of trigger is useful for enforcing policies:

- If any trigger action returns a nonzero exit status, the operation is canceled.
- If all trigger actions return a zero exit status, the operation proceeds.

For example, a preoperation trigger can prohibit checkin of an element that fails to pass a code-quality test.

A postoperation trigger (**–postop** option) fires after completion of the corresponding operation. The one or more actions you've specified take place in their order on the command line. This kind of trigger is useful for recording—in the VOB or outside it—the occurrence of the operation. If a postoperation trigger action returns a nonzero exit status, ClearCase, ClearCase LT and Attache display a `failed exit status` warning message, but continues to perform other trigger actions, if any.

For example, a post-operation trigger on **checkin** attaches an attribute to the checked-in version and sends a mail message to interested users and/or managers.

RESTRICTION LISTS AND INCLUSION LISTS

You can define an element trigger type or all-element trigger type with a *restriction list*, which limits the scope of the operation specified with **–preop** or **–postop**. The trigger fires only if the operation involves particular type objects.

A type trigger type is not associated with element objects, but with one or more type objects. When creating a type trigger type, you must specify an *inclusion list*, naming the type objects to be associated with the new trigger type. (Hence, it is unnecessary to use **mktrigger** to create the association.) The special keyword **–all** allows you to associate a type trigger type with *every* type object of a particular kind (for example, all branch type objects), even those objects created after you enter this command.

TRIGGER ENVIRONMENT VARIABLES

When a trigger fires, the trigger action executes in a special environment whose EVs make information available to **–exec**, **–execunix**, and **–execwin** routines: what operation caused the

trigger to fire, what object was involved in the operation, and so on. The complete set of EVs is listed in *TRIGGER OPERATIONS AND TRIGGER ENVIRONMENT VARIABLES on page 668*.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type owner (applies to **–replace** only), VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, trigger type (applies to **–replace** only).

OPTIONS AND ARGUMENTS

SPECIFYING THE KIND OF TRIGGER TYPE. *Default:* None.

–ele·ment
Creates an *element trigger type*, which can be attached to individual elements with **mktrigger**.

–ele·ment –a·ll
Creates an *all-element trigger type*, which is effectively attached to the entire VOB.

–typ·e
Creates a *type trigger type*, and associates it with specific type objects and/or kinds of type objects.

HANDLING OF NAME COLLISIONS. *Default:* An error occurs if a trigger type named *type-name* already exists in the VOB.

–rep·lace
Replaces the existing definition of *type-name* with a new one. If you do not include options from the existing definition, their values are replaced with the defaults.

If you specify a comment when using **–replace**, the comment appears in the event record for the modification (displayed with **lshistory –minor**); it does not replace the object's creation comment (displayed with **describe**). To change an object's creation comment, use **chevent**.

Constraint: If an instance of an element trigger type is currently attached to any element, the replacement definition must also be of an element trigger type (but not an all-element trigger type). You can remove an existing trigger type and all of its attached instances using the **rmtype** command.

SPECIFYING THE OPERATIONS TO BE MONITORED. *Default:* None.

–pre·op *opkind*[,...]
Specifies one or more operations that cause the trigger to fire before the operation starts. The exit status of the trigger actions is significant: for each trigger action, a zero exit status allows the operation to proceed; a nonzero exit status cancels the operation.

–pos·top *opkind*[,...]
Specifies one or more operations that cause the trigger to fire after the operation completes. The exit status of the trigger action is not significant.

For both **–preop** and **–postop**, you must specify a comma-separated list of operations, any of which fire the trigger. Many of the operation keywords have the same names as **cleartool** subcommands (for example, **checkout** and **unlock**). Uppercase keywords (for example, MODIFY_ELEM) identify groups of operations. See the *TRIGGER OPERATIONS AND TRIGGER ENVIRONMENT VARIABLES* section for a list of operation keywords.

SUPPRESSING TRIGGER FIRING FOR CERTAIN USERS. *Default:* Triggers fire regardless of who performs the operation.

–nus·ers *login-name*[,...]
Suppresses trigger firing when any user on the comma-separated *login-name* list performs the operation.

SPECIFYING THE TRIGGER ACTION. *Default:* None. Specify one or more of the following options to indicate the action to be performed when the trigger fires; you can use more than one option of the same kind. With multiple options, the trigger actions are performed in the specified sequence.

–exe·c *command*
Executes the specified command in a Bourne shell when the trigger fires. If *command* includes one or more arguments, quote the entire string. Use single quotes ('*command* ') if the command includes ClearCase, ClearCase LT, or Attache environment variables, to delay interpretation until trigger firing time.

–execu·nix *command*
–execw·in *command*
These options have the same behavior as **–exec** when fired on the appropriate platform (UNIX or Windows, respectively). When fired on the other platform, they do nothing; however, triggers with only **–execunix** commands always fail on Windows, and triggers that only have **–execwin** commands always fail on UNIX.

NOTE: If you use **–execwin** when defining a trigger type on UNIX, you must escape backslashes (\ ) in *command* with a backslash. Also, if you invoke a command built in to the Windows shell (for example, **cd**, **del**, **dir**, or **copy**), you must invoke the shell with **cmd /c**. For example:

–execwin 'cmd /c copy %CLEARCASE_PN% %HOME%'

–mkl·abel *label-type-selector*
(With **–postop** only) Attaches the specified version label to the version involved in the operation that caused trigger firing. If the label type is a global type, a local copy of the

type must exist in the VOB in which you are creating the trigger type. Specify *label-type-selector* in the form [**lbtype:**]*type-name*[**@***vob-selector*]

type-name Name of the label type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–mka·**ttr** *attribute-type-selector***=***value*

(With **–postop** only) Attaches the specified attribute name/value pair to the object involved in the operation that caused trigger firing. If the attribute type is a global type, a local copy of the type must exist in the VOB in which you are creating the trigger type. Specify *attribute-type-selector* in the form [**attype:**]*type-name*[**@***vob-selector*]

type-name Name of the attribute type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–mkh·**link** *hlink-type-selector***,to=***pname*

(With **–postop** only) Creates a hyperlink from the object involved in the operation that caused the trigger to fire to the object specified by *pname*. If the hyperlink type is a global type, a local copy of the type must exist in the VOB in which you are creating the trigger type. Specify *hlink-type-selector* in the form [**hltype:**]*type-name*[**@***vob-selector*]

type-name Name of the hyperlink type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–mkh·**link** *hlink-type-selector*,**from=***pname*

(With **–postop** only) Creates a hyperlink from the object specified by *pname* to the object involved in the operation that caused the trigger to fire. If the hyperlink type is a global type, a local copy of the type must exist in the VOB in which you are creating the trigger type. Specify *hlink-type-selector* in the form [**hltype:**]*type-name*[**@***vob-selector*]

type-name Name of the hyperlink type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

NOTES: With the built-in actions **–mklabel**, **–mkattr**, and **–mkhlink**, you can specify the information either literally or using environment variables:

–mklabel RLS_2.3	*(literal)*
–mklabel RLS_$RLSNUM	*(depends on value of EV at trigger firing time)*
–mklabel $THIS_RLS	*(depends on value of EV at trigger firing time)*
–mkattr ECO=437	*(literal)*
–mkattr ECO=$ECONUM	*(depends on value of EV at trigger firing time)*

The built-in actions never cause additional triggers to fire. However, scripts invoked with **–exec** may cause such chain reactions. For example, a **mklabel** command in a shell script can cause another trigger to fire, but the corresponding **–mklabel** trigger action cannot.

ELEMENT TRIGGER TYPES: SPECIFYING A RESTRICTION LIST. *Default:* No restrictions; triggers fire when any of the specified operations occurs, no matter what type objects are involved.

–att·ype *attr-type-selector*[,...]
–brt·ype *branch-type-selector*[,...]
–elt·ype *elem-type-selector*[,...]
–hlt·ype *hlink-type-selector*[,...]
–lbt·ype *label-type-selector*[,...]
–trt·ype *trigger-type-selector*[,...]

Use one or more of the above options (or multiple options of the same kind) to specify a set of type objects for the *restriction list*. If the type object is an ordinary type, it must already exist. If a type object is a global type and a local copy does not exist in the VOB, a local copy is created automatically.

Repeated options, such as **–elt text_file –elt c_source**, are equivalent to a single option: **–elt text_file,c_source**. Wildcarding (**–eltype '*file'**) is not supported.

At trigger firing time, the items on the restriction list form a logical condition. If the condition is met, the trigger fires.

Specify the *type-selector* arguments in the form [*type-kind***:**]*type-name*[**@***vob-selector*]

type-kind — One of

attype	attribute type
brtype	branch type
eltype	element type
hltype	hyperlink type
lbtype	label type
trtype	trigger type

type-name — Name of the type object

vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

NOTE: Suppressing the firing of a preoperation trigger means that the operation is allowed to proceed.

Here is a simple condition:

–brtype rel2_bugfix — Fire the trigger only if the operation involves a branch of type **rel2_bugfix**.

If the list includes multiple type objects, they are combined into a compound condition: type objects of the same kind are grouped with logical OR; objects (or groups) of different kinds are then logically ANDed.

–brtype rel2_bugfix –eltype text_file,c_source	Fire the trigger only if the operation involves a branch of type **rel2_bugfix** AND it involves either an element of type **text_file** OR of an element of type **c_source**.

In forming the condition, a type object is ignored if it could not possibly be affected by the operation. (The relevant information is included in the *TRIGGER OPERATIONS AND TRIGGER ENVIRONMENT VARIABLES* section.) For example, the restriction list **–lbtype REL2,REL2.01** applies only to the operations **chtype**, **mklabel**, and **rmlabel**.

TYPE TRIGGER TYPES: SPECIFYING AN INCLUSION LIST. *Default:* None. You must specify at least one item for the inclusion list of a type trigger type.

–att·ype *attr-type-selector*[,...]	or	**–att·ype -all**
–brt·ype *branch-type-selector*[,...]	or	**–brt·ype -all**
–elt·ype *elem-type-selector*[,...]	or	**–elt·ype -all**
–hlt·ype *hlink-type-selector*[,...]	or	**–hlt·ype -all**
–lbt·ype *label-type-selector*[,...]	or	**–lbt·ype -all**
–trt·ype *trigger-type-selector*[,...]	or	**–trt·ype -all**

You must specify at least one existing type object, or at least one kind of type object, using the special keyword **–all**. The trigger fires only if the inclusion list contains the type object that is being modified or used by the operation.

TRACING TRIGGER EXECUTION. *Default:* At trigger firing time, if environment variable **CLEARCASE_TRACE_TRIGGERS** is set to a nonnull value in the process that causes the trigger to fire, a message that includes the trigger type name is sent to **stdout** when the trigger fires; a similar message is generated when the trigger action completes.

–pri·nt
Causes the messages to be generated at trigger firing time, whether or not **CLEARCASE_TRACE_TRIGGERS** is set.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

NAMING THE TRIGGER TYPE. *Default:* The trigger type is created in the VOB that contains the current working directory unless you use the **@***vob-selector* suffix to specify another VOB.

type-selector ...

One or more names for the trigger types to be created. Specify *trigger-type-selector* in the form [**trtype:**]*type-name*[**@***vob-selector*]

type-name — Name of the trigger type

See the *Object Names* section in the **cleartool** reference page for rules about composing names.

vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

TRIGGER OPERATIONS AND TRIGGER ENVIRONMENT VARIABLES

Trigger Operations for Type Trigger Types

The following list shows the operation keywords (**opkind**) for use in definitions of type trigger types (**mktrtype –type**). The operation fires a trigger only if the affected object is a type object specified on the inclusion list. (This list is required.)

NOTE: These operations are not ClearCase or ClearCase LT commands, although some have the same names as **cleartool** subcommands. These are lower-level operations, similar to function calls. See the **events_ccase** reference page for a list of which commands cause which operations.

MODIFY_TYPE

mktype (see following **NOTE)**
rmtype
rntype
lock
unlock
chevent
chmaster

NOTE: If you specify **mktype**, the corresponding inclusion list cannot specify individual type objects; all relevant options must use the **–all** keyword. For example:

... –postop mktype –eltype –all –brtype -all ...

Trigger Operations for Element and All-Element Trigger Types

Table 8 lists the operation keywords (**opkind**) for use in definitions of element and all-element trigger types (**–element** and **–element –all**). See also the **events_ccase** reference page.

NOTE: These operations are not ClearCase or ClearCase LT commands, although some have the same names as **cleartool** subcommands. These are lower-level operations, similar to function calls. See the **events_ccase** reference page for a list of which commands cause which operations.

Table 8 Trigger Definition Operation Keywords

Operation Keyword	Restrictions Checked when Trigger Fires
MODIFY_ELEM	
checkout	Element type, branch type
reserve	Element type, branch type
uncheckout	Element type, branch type
unreserve	Element type, branch type
MODIFY_DATA	
checkin	Element type, branch type
chevent	See **NOTE** at end of table
chtype	All type objects
lnname	Element type, branch type
lock	See **NOTE** at end of table
mkbranch	Element type, branch type
mkelem	Element type
mkslink	N/A
protect	See **NOTE** at end of table
rmbranch	Element type, branch type
rmelem	Element type
rmname	N/A
rmver	Element type, branch type
unlock	See **NOTE** at end of table
MODIFY_MD	

Table 8 Trigger Definition Operation Keywords

Operation Keyword	Restrictions Checked when Trigger Fires
chevent	see **NOTE** at end of table
chmaster	See **NOTE** at end of table
mkattr	Element type, attribute type, branch type
mkhlink	Element type, hyperlink type, branch type
mklabel	Element type, label type, branch type
mktrigger	Element type, trigger type
rmattr	Element type, attribute type, branch type
rmhlink	Element type, hyperlink type, branch type
rmlabel	Element type, label type
rmtrigger	Element type, trigger type

NOTE: The operation fires a trigger only if the affected object is one of the following:

- A branch object or version object (in this case, only element type and branch type restrictions apply)
- An element object (in this case, only element type restrictions apply)
- A type object (in this case, only restrictions on that kind of type object apply)

Trigger Environment Variables

The following list shows the EVs that are set in the environment in which a trigger action program runs. The words in parentheses at the beginning of the description indicate which operations cause the EV to be set to a significant string; for all other operations, the EV is set to the null string. (See the **events_ccase** reference page for a list of which commands cause which operations.)

CLEARCASE_ATTACH

(**mktrigger**, **rmtrigger**) Set to 1 if an element trigger type (except an *all-element trigger type*) is on the affected element's *attached list*; set to 0 if it is on a directory element's *inheritance list*. See the **mktrigger** reference page for a description of these lists.

CLEARCASE_ATTYPE
(All operations that can be restricted by attribute type) Attribute type involved in operation that caused the trigger to fire. In a **rename** operation, the old name of the renamed attribute type object.

CLEARCASE_BRTYPE
(All operations that can be restricted by branch type) Branch type involved in the operation that caused the trigger to fire. In a **rename** operation, the old name of the renamed branch type object.

CLEARCASE_CHGRP
(**protect**) New group of the reprotected object as specified in the command line; unset if not specified.

CLEARCASE_CHMOD
(**protect**) New protection of the reprotected object as specified in the command line; unset if not specified.

CLEARCASE_CHOWN
(**protect**) New owner of the reprotected object as specified in the command line; unset if not specified.

CLEARCASE_CI_FPN
(**checkin**) Pathname in **checkin –from**.

CLEARCASE_CMDLINE
(All operations initiated through use of the **cleartool** command) A string specifying the **cleartool** subcommand and any options and arguments included on the command line.

NOTES:

- This EV's value is set by the **cleartool** command, and only by that command. If a trigger is fired by any other means (through the use of a ClearCase or ClearCase LT GUI, for example) the EV is not set.
- The EV's value may be garbled if the command line contains nested quotes.

CLEARCASE_COMMENT
(All operation kinds that support comments) Comment string for the command that caused the trigger to fire.

CLEARCASE_ELTYPE
(All operations that can be restricted by element type) Element type of the element involved in the operation that caused the trigger to fire. In a **rename** operation, the old name of the renamed element type object.

CLEARCASE_FREPLICA
(**chmaster**) The old master replica, or "from-replica": the replica that mastered the object at the time the command was entered.

When the command **chmaster –default brtype:***branch-type-name* is run at the site of the replica that masters the branch type, **CLEARCASE_FREPLICA** is set to the name of the current replica. If the command is run at a site that does not master the branch type, the command fails, but **CLEARCASE_FREPLICA** is set to the name of the replica that masters the branch type.

When the command **chmaster –default** *branch-name* is run, **CLEARCASE_FREPLICA** is set to the name of the current replica. (If the command is run at a site that does not master the branch, it fails.)

CLEARCASE_FTEXT
(**mkhlink**, **rmhlink**) Text associated with hyperlink from-object.

CLEARCASE_FVOB_PN
(**mkhlink**, **rmhlink**) Pathname of VOB containing hyperlink from-object.

CLEARCASE_FXPN
(**mkhlink**, **rmhlink**) VOB-extended pathname of hyperlink from-object.

CLEARCASE_HLTYPE
(All operations that can be restricted by hyperlink type) Hyperlink type involved in operation that caused the trigger to fire. In a rename operation, the old name of the renamed hyperlink type object.

CLEARCASE_ID_STR
(**checkin**, **checkout**, **mkattr**, **mkbranch**, **mkhlink**, **mklabel**, **rmattr**, **rmhlink**, **rmlabel**, **rmver**) *Version-ID* of version, or *branch pathname* of branch, involved in the operation.

CLEARCASE_IS_FROM
(**mkhlink**, **rmhlink**) Set to 1 if **CLEARCASE_PN** contains name of hyperlink from-object; set to 0 if **CLEARCASE_PN** contains name of hyperlink to-object.

CLEARCASE_LBTYPE
(All operations that can be restricted by label type) Label type involved in the operation that caused the trigger to fire. In a **rename** operation, the old name of the renamed label type object.

CLEARCASE_MTYPE
(All) Kind of object involved in the operation that caused the trigger to fire: element type, branch type, directory version, and so on.

CLEARCASE_NEW_TYPE
(**rename**) New name of the renamed type object.

CLEARCASE_OP_KIND
(All) Actual operation that caused the trigger to fire.

CLEARCASE_OUT_PN
(**checkout**) Pathname in **checkout –out**. (Same as **CLEARCASE_PN** if **–out** not used.)

CLEARCASE_PN
(All operations; element triggers only) Name of element specified in the command that caused the trigger to fire.

NOTES:

- With an all-element trigger, a pathname in the root directory of a VOB is reported with an extra (but still correct) "/." pathname component:
  `/vobs/proj/./releasedir` *(if VOB is mounted at '/vobs/proj')*
- Some **cleartool** and Attache commands rename files during their execution. Usually, such manipulations are unnoticeable, but you may need to adjust your trigger scripts accordingly. For example, the script for a preoperation **mkelem** trigger may need to operate on file name "$**CLEARCASE_PN**.mkelem" instead of "$**CLEARCASE_PN**".
- If the file does not exist (for example, the checked-out file was removed), the value of **CLEARCASE_PN** is different from its value when the file exists.

CLEARCASE_PN2
(**lnname**)

- When a side-effect of a **mkelem** operation, gets the same value as **CLEARCASE_PN**.
- When a side-effect of a **mv** operation, gets the old pathname of the element.

CLEARCASE_POP_KIND
(**mkelem**, **mkslink**, **lnname**, **rmname**) Parent operation kind. The **mkelem** and **mkslink** operations both cause an **lnname** operation. If **lnname** happens as a result of either of these parent operations, **CLEARCASE_POP_KIND** is set to **mkelem** or **mkslink**, respectively. Note that both the parent operations (**mkelem** and **mkslink**) and the child operation (**lnname**) set **CLEARCASE_POP_KIND** to the applicable parent operation value—**mkelem** or **mkslink**.

User Commands that Cause Multiple Operations	Operations	CLEARCASE_POP_KIND value
mkelem	**mkelem**	**mkelem**
	lnname	**mkelem**
ln –s	**mkslink**	**mkslink**
	lnname	**mkslink**

User Commands that Cause Multiple Operations	Operations	CLEARCASE_POP_KIND value
move | **mv**	**lnname**	**rmname**
	rmname	**lnname**

The **move** or **mv** command is a special case because there is no **move** operation. Therefore, the **CLEARCASE_POP_KIND** environment variable is set to the values **rmname** and **lnname** to show that those operations were part of the command execution.

CLEARCASE_PPID
(All) Parent Process-ID: the process-ID of the ClearCase or ClearCase LT program (for example, **cleartool**) that invoked the trigger. This is useful for constructing unique names for temporary files that will pass data between a preoperation trigger and a postoperation trigger, or between successive parts of a multipart trigger action. **CLEARCASE_PPID** is not useful for Attache clients.

CLEARCASE_RESERVED
(**checkin**, **checkout**) Set to 1 if user requested a reserved checkout; set to 0 if user requested an unreserved checkout.

CLEARCASE_SLNKTXT
(**mkslink**; that is, the **ln –s** command) Text of the new VOB symbolic link.

CLEARCASE_TREPLICA
(**chmaster**) The new master replica, or "to-replica": the replica specified to receive mastership.

When the command **chmaster –default brtype:***branch-type-name* is run at the site of the replica that masters the branch type, **CLEARCASE_TREPLICA** is set to the name of the current replica. If the command is run at a site that does not master the branch type, the command fails, but **CLEARCASE_TREPLICA** is set to the name of the current replica.

When the command **chmaster –default** *branch-name* is run, **CLEARCASE_TREPLICA** is set to the name of the replica that masters the branch type. (If the command is run at a site that does not master the branch, it fails.)

CLEARCASE_TRTYPE
(All operations that can be restricted by trigger type) Trigger type involved in the operation that caused the trigger to fire. In a **rename** operation, the old name of the renamed trigger type object.

CLEARCASE_TTEXT
(**mkhlink**, **rmhlink**) Text associated with hyperlink to-object.

CLEARCASE_TVOB_PN
(**mkhlink**, **rmhlink**) Pathname of VOB containing hyperlink to-object.

CLEARCASE_TXPN
(**mkhlink**, **rmhlink**) VOB-extended pathname of hyperlink to-object.

CLEARCASE_USER
(All) The user who issued the command that caused the trigger to fire; derived from the UNIX-level real user-ID.

CLEARCASE_VAL
(**mkattr**) String representation of attribute value for **CLEARCASE_ATTYPE** (for example, `"Yes"` or `4657`).

CLEARCASE_VIEW_TAG
(All) View-tag of the view in which the operation that caused the trigger to fire took place.

CLEARCASE_VOB_PN
(All) VOB-tag of the VOB whose object was involved in the operation that caused the trigger to fire.

CLEARCASE_VTYPE
(**mkattr**) Value type of the attribute in **CLEARCASE_ATTYPE** (for example, string or integer).

CLEARCASE_XN_SFX
(All) Extended naming symbol (such as @@) for host on which the operation took place.

CLEARCASE_XPN
(All operations; element triggers only) Same as **CLEARCASE_ID_STR**, but prepended with **CLEARCASE_PN** and **CLEARCASE_XN_SFX** values, to form a complete VOB-extended pathname of the object involved in the operation.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

Trigger environment variables are typically evaluated when the trigger fires, not when you enter the **mktrtype** command. If this is the case, escape the dollar sign (**$**), either by enclosing it with single quotes or by preceding it with a backslash (\). This escaping is not necessary if you enter the command manually in **cleartool**'s interactive mode (that is, if it is not interpreted by a shell).

- Create an element type named **script** for use with shell-script files. Then, create an all-element trigger type, **chmod_a_plus_x**, that makes newlycreated elements of type **script** executable. Convert a view-private file to an element of this type.

cmd-context **mkeltype -supertype text_file -c "shell script" script**
```
Created element type "script".
```

cmd-context **mktrtype -element -all -postop mkelem -eltype script -nc **
-exec '/usr/atria/bin/cleartool protect -chmod a+x $CLEARCASE_PN' chmod_a_plus_x
```
Created trigger type "chmod_a_plus_x".
```

cmd-context **mkelem -eltype script -ci -nc cleanup.sh**
```
Created element "cleanup.sh" (type "script").
Changed protection on "/usr/hw/src/cleanup.sh".
Checked in "cleanup.sh" version "/main/1".
```

- Create an all-element trigger type, to prevent files with certain extensions from being made into elements.

  cmd-context **mktrtype -element -all -nc -preop mkelem **
  -exec /usr/local/bin/check_ext check_ext

  ```
  Created trigger type “check_ext”.
  ```

- Create an all-element trigger type, to run a script each time a checkin takes place.

  cmd-context **mktrtype -element -all -postop checkin -nc **
  -exec /usr/local/bin/notify notify_admin
  ```
  Created trigger type "notify_admin".
  ```

  ```
  ‘notify’ script:
  mail jones adm <<!
  "notify_admin" Trigger:
  checkin of "$CLEARCASE_PN"
  version: $CLEARCASE_ID_STR
  by: $CLEARCASE_USER
  comment:
  $CLEARCASE_COMMENT
  !
  ```

- Create an element trigger type that runs a script when a **mkbranch** command is executed. Specify different scripts for UNIX and Windows platforms.

  cmd-context **mktrtype –element –postop mkbranch –nc **
  **–execunix /net/neon/scripts/branch_log.sh **
  –execwin \\\\photon\\triggers\\branch_log.bat branch_log
  ```
  Created trigger type "branch_log".
  ```

- Create an all-element trigger type to monitor checkins of elements of type **c_source**. Firing the trigger runs a test program on the file being checked in, and may cancel the checkin.

cmd-context **mktrtype -element -all -nc -preop checkin **
**-exec '$CLEARCASE_VOB_PN/scripts/metrics_test $CLEARCASE_PN' **
-eltype c_source metrics_trigger
```
Created trigger type "metrics_trigger".
```

Use of environment variable **CLEARCASE_VOB_PN** causes the test program to be retrieved from a location in the current VOB.

- Create an all-element trigger type to attach a version label to each new version created on any element's **main** branch.

  cmd-context **mktrtype -element -all -postop checkin -mklabel REL\$BL_NUM **
  -nc -brtype main label_i
  ```
  Created trigger type "label_it".
  ```

  Environment variable **BL_NUM** determines which version label is to be attached. This EV is evaluated at trigger firing time, because the dollar sign ($) is escaped.

- Create a type trigger type to send a mail message each time any new branch type is created.

  cmd-context **mktrtype -type -nc -postop mktype -brtype -all **
  -exec '$CLEARCASE_VOB_PN/scripts/mail_admin' new_branch_trigger
  ```
  Created trigger type "new_branch_trigger".
  ```

- Create a type trigger type to monitor the creation of new label types. The trigger script aborts the label-type-creation operation if the specified name does not conform to standards.

  cmd-context **mktrtype -type -nc -preop mktype -lbtype -all **
  -exec '$CLEARCASE_VOB_PN/scripts/check_label_name' check_label_trigger
  ```
  Created trigger type "check_label_trigger".
  ```

- Create an element trigger type that, when attached to an element, fires whenever a new version of that element is checked in. Firing the trigger attaches attribute **TestedBy** to the version, assigning it the value of the **CLEARCASE_USER** environment variable as a double-quoted string.

  NOTE: In this example, the single quotes preserve the double quotes on the string literal, and suppress environment variable substitution by the shell. The **CLEARCASE_USER** environment variable is evaluated at firing time.

  cmd-context **mktrtype -element -postop checkin **
  **-c "set attribute to record which user checked in this version" **
  -mkattr 'TestedBy="$CLEARCASE_USER"' trig_who_didit
  ```
  Created trigger type "trig_who_didit".
  ```

- Create an all-element trigger type that prompts for the source of an algorithm when an element of type **c_source** is created. Firing the trigger executes a script named **hlink_algorithm**, which invokes the **clearprompt** utility to obtain the necessary

information. The script then creates a text-only hyperlink between the newly created element object (for example, **foo.c@@**) and the specified text. The **hlink_algorithm** script is shown immediately after the **mktrtype** command.

cmd-context **mktrtype -element -all -nc -postop mkelem -eltype c_source \ -exec '$CLEARCASE_VOB_PN/scripts/hlink_algorithm' describe_algorithm**

```
Created trigger type "describe_algorithm".

hlink_algorithm script:

clearprompt text -outfile /usr/tmp/alg.$CLEARCASE_PPID -multi_line \
-def "Internal Design" -prompt "Algorithm Source Document:"

TOTEXT='cat /usr/tmp/alg.$CLEARCASE_PPID'
cleartool mkhlink -ttext "$TOTEXT" design_spec
$CLEARCASE_PN$CLEARCASE_XN_SFX

rm /usr/tmp/alg.$CLEARCASE_PPID
```

- Use a postoperation trigger to modify the user-supplied comment whenever a new version is created of an element of type **header-file**

  cmd-context **mktrtype -element -all -nc -postop checkin -eltype header_file \ -exec '/usr/local/scripts/hdr_comment' change_header_file_comment**

```
Created trigger type "change_header_file_comment".

hdr_comment script:

# analyze change to header file
CMNT='/usr/local/bin/analyze_hdr_file $CLEARCASE_PN'

# append analysis to user-supplied checkin comment
cleartool chevent -append -c "$CMNT" $CLEARCASE_PN'
```

- Create an all-element trigger type and a type trigger type that prevent all users except **stephen**, **hugh**, and **emma** from running the **chmaster** command on element-related objects and type objects in the current VOB:

  cleartool mktrtype –element –all –preop chmaster –nusers stephen,hugh,emma \ –execunix 'Perl –e "exit –1;"' –execwin 'ccperl –e "exit (–1);"' \ –c "ACL for chmaster" elem_chmaster_ACL

  cleartool mktrtype –type –preop chmaster –nusers stephen,hugh,emma \ –execunix 'Perl –e "exit –1;"' –execwin 'ccperl –e "exit (–1);"' \ –attype –all –brtype –all –eltype –all –lbtype –all –hltype –all \ –c "ACL for chmaster" type_chmaster_ACL

SEE ALSO

events_ccase, **lstype**, **mktrigger**, **rmtype**, **type_object**

mkview

Creates and registers a view

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only—Create and register a dynamic view:

  mkview –tag *dynamic-view-tag* [**–tco·mment** *tag-comment*]
  [**–tmo·de** { **insert_cr** | **transparent** | **strip_cr** }]
  [**–reg·ion** *network-region*] [**–ln** *remote-storage-dir-pname*]
  [**–nca·exported**] [**–cac·hesize** *size*]
  [**–sha·reable_dos** | **–nsh·areable_dos**] [**–str·eam** *stream-selector*]
  { **–stg·loc** { *view-stgloc-name* | **–aut·o** }
  | [**–hos·t** *hostname* **–hpa·th** *host-storage-pname* **–gpa·th** *global-storage-pname*]
  dynamic-view-storage-pname }

- ClearCase and Attache only—Create and register a snapshot view:

  mkview –sna·pshot [**–tag** *snapshot-view-tag*] [**–tco·mment** *tag-comment*]
  [**–tmo·de** { **insert_cr** | **transparent** | **strip_cr** }]
  [**–cac·hesize** *size*] [**–pti·me**] [**–str·eam** *stream-selector*]
  [**–stg·loc** *view-stgloc-name*
  | **–col·ocated_server** [**–hos·t** *hostname* **–hpa·th** *host-snapshot-view-pname*
  –gpa·th *global-snapshot-view-pname*]] *snapshot-view-pname*

- ClearCase LT only—Create and register a snapshot view:

  mkview [**–sna·pshot**] [**–tag** *view-tag*] [**–tco·mment** *tag-comment*]
  [**–tmo·de** { **insert_cr** | **transparent** | **strip_cr** }]
  [**–pti·me**] [**–str·eam** *stream-selector*]
  [**–stg·loc** *view-stgloc-name*] *snapshot-view-pname*

DESCRIPTION

The **mkview** command creates a new view as follows:

- Creates a view storage directory at a specified network location or in a server storage location previously created with the **mkstgloc** command. In ClearCase LT, these locations are always on the ClearCase LT server host. The view storage directory maintains information about the view. Along with other files and directories, the directory contains the view's config spec and the view database. See the **view** reference page for details.
- Creates a view-tag, the name by which users access a dynamic view. Snapshot views also have view-tags, but users access snapshot views by setting the snapshot view directory, as with **cd**.
- For a snapshot view, creates the snapshot view directory. This is the directory into which your files are loaded when you populate the view using **update**.
- Places entries in the network's view registry; use the **lsview** command to list view tags.
- Starts a **view_server** process on the host where the view storage directory physically resides. The **view_server** process manages activity in a particular view. It communicates with VOBs during **checkout**, **checkin**, **update**, and other operations.

Interop Text Modes

Operating systems use different character sequences to terminate lines of text files. In UNIX, the line terminator for text files is a single **<LF>** character. On Windows systems, the standard line terminator is **<CR><LF>**. Each view has an interop text mode—specified by the **–tmode** option—that determines the line terminator sequence for text files in that view. The interop text mode also determines whether line terminators are adjusted before a text file is presented to the view (at checkout time, for example). For example, a text file element created by a Windows client that is accessed through a UNIX view would be stripped of **<CR>** characters, and the **<CR>** characters would be reinserted when the file was written to the VOB as a new version.

In Attache, when you use **mkws** to create a workspace, you can create an associated view at the same time. The **mkws** command does not take the **–tmode** option, but the Attache client has a preference you can set to specify the interop text mode for any views created on behalf of a workspace.

For more information, see *Administering ClearCase* and the reference pages for **msdostext_mode** and **mkeltype**.

Views and UCM Streams

Views are attached to streams in the UCM model. Only views can modify a UCM stream. Views cannot be moved between streams or detached from a stream without removing the view.

ClearCase and Attache Dynamic Views Only—Marking a View for Export

A dynamic view to be used for NFS export of one or more VOBs (for access by applications other than those in the ClearCase Product Family) must be marked in the registry as an export view. Each export view is assigned an export-ID, which ensures that NFS-exported view/VOB

combinations have stable NFS file handles across server reboots or shutdown and restart of ClearCase.

If the dynamic view is registered in multiple regions, the export marking must be on the view-tag in the server host's default region. To create an export view, use the **–ncaexported** option. You can register an existing dynamic view or VOB for export by using **mktag –replace –ncaexported**. For information on exporting view-VOB combinations, see the **export_mvfs** reference page.

Setting the Cache Size for Views

Although both kinds of views use caches, cache size is more significant for a dynamic view than it is for a snapshot view. The dynamic view's cache size determines the number of VOB lookups that can be stored. You can set the size of the cache with the **–cachesize** option. This creates the following line in the **.view** file for the view:

–cache *size*

When a **view_server** process is started, it uses this value. For more information on the **view_server** cache and changing its size, see the **view_server**, **setcache**, and **chview** reference pages.

ClearCase and Attache Dynamic Views Only—Using Express Builds

You can configure a dynamic view to use the express builds feature by creating the view with the **–nshareable_dos** option. When you invoke **clearmake** in this kind of view, **clearmake** builds nonshareable derived objects (DOs). Information about these DOs is not written into the VOB, so the build is faster; however, nonshareable DOs cannot be winked in by other views.

If you do not specify **–sha·reable_dos** or **–nsh·areable_dos**, **mkview** uses the site-wide default set in the registry (with the **setsite** command). If there is no site-wide default, **mkview** configures the view so that builds in the view create shareable DOs.

To change the DO property for an existing view, use the **chview** command. For more information on shareable and nonshareable DOs, see *Building Software with ClearCase*.

ClearCase and Attache Dynamic Views Only—Activating a View

Creating a view-tag also executes the **startview** command, which activates the dynamic view on the current host (unless the tag's target network region does not include the local host.) It also places an entry in the host's viewroot directory. (For example, specifying **–tag gamma** creates the entry **/view/gamma**.)

After it is activated, a dynamic view can be set with the **setview** command; it can also be accessed with view-extended naming. (For details, see the **startview**, **view**, and **pathnames_ccase** reference pages.)

ClearCase, Attache, and ClearCase LT Snapshot Views Only—Activating a View

Snapshot views cannot be explicitly activated and cannot be accessed with view-extended naming. However, a snapshot view becomes active when you change to the view directory and issue a ClearCase or ClearCase LT command.

View Creator Identity and umask Permissions

Avoid creating views as **root**. This often causes problems with remote access to a view, because **root** on one host often becomes user-ID **–2** when accessing other hosts.

Your current **umask(1)** setting determines which users can access the view. For example, a umask value of 2 allows anyone to read data in the view, but only you (the view's owner) and others in your group can write data to it—create view-private files, build derived objects, and so on. If your umask value is 22, only you can write data to the new view.

Reconfiguring a View

A view's associated **view_server** process reads a configuration file when it starts up. You can revise this file—for example, to make the view read-only. See the **view_server** reference page for details.

Backing Up a View

For information about performing view backups, see *Administering ClearCase*.

If you create a snapshot view in which the view-storage directory is located outside the snapshot view, you must back up recursively both the view storage directory and the snapshot view's root directory.

Deleting a View

The view created by this command is the root of a standard directory tree; but a view must be deleted only with the **rmview** command, never with standard **rm(1)**. See the **rmview** reference page for details.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE VIEW-TAG. *Default for ClearCase and Attache dynamic views:* None. *Default for ClearCase LT and ClearCase/Attache snapshot views:* A generated tag.

–tag *view-tag*

Dynamic view—Specifies a name for the view, in the form of a simple file name. This name appears in the local host's file system as a subdirectory of **/view**, the viewroot directory. For example, the view **experiment** appears as **/view/experiment**.

Snapshot view—Specifies a name for the view as it is recorded in the registry.

ClearCase and Attache only—If your network has multiple regions, use the **mktag** command to create an additional view-tag for each additional region.

–tco·mment *tag-comment*
Adds a comment to the view-tag's entry in the **view_tag** registry. Use **lsview –long** to display the tag comment.

SPECIFYING THE KIND OF VIEW. *Default for ClearCase and Attache:* Dynamic view. *Default for ClearCase LT:* **–snapshot** (the ClearCase LT synopsis for this command retains this option, even though it is the default, for easier migration of view-creation scripts from ClearCase LT to ClearCase).

–sna·pshot
Specifies a snapshot view. See the **view** reference page for a discussion of views and the differences between snapshot and dynamic views.

SPECIFYING THE INTEROP TEXT MODE. *Default*: **–tmode transparent** for views created on UNIX machines or those created through the MSDOS command line. **–tmode transparent** is also the default for views created through the Windows GUI unless a different site-wide interop text mode has been set with **setsite**.

NOTE: VOBs that are to be accessed by interop text mode views must be enabled to support such views. See the **vob** and **msdostext_mode** reference pages.

–tmo·de transparent
A **transparent** interop text mode view is created. The line terminator for text files is a single **<NL>** character. The view does not transform text file line terminators in any way.

–tmo·de insert_cr
Creates an **insert_cr** interop text mode view. The view converts **<NL>** line terminators to the **<CR><NL>** sequence when reading from a VOB, and **<CR><NL>** line terminators to single **<NL>** characters when writing to the VOB.

–tmo·de strip_cr
Creates a **strip_cr** interop text mode view. The view converts **<CR><NL>** line terminators to **<NL>** when reading from a VOB, and **<NL>** line terminators back to the **<CR><NL>** sequence when writing to the VOB.

SPECIFYING A NETWORK REGION. *Default:* The local host's network region, as listed by the **hostinfo –long** command. See the **registry_ccase** reference page for a discussion of network regions.

–reg·ion *network-region*
Creates the view-tag in the specified network region. An error occurs if the region does not already exist.

CAUTION: The view-tag created with **mkview** must be for the network region to which the view server host belongs. Thus, use this option only when you are logged in to a remote host that is in another region. Moreover, a view-tag for the view's home region must always exist.

REMOTE PRIVATE STORAGE AREA. *Default:* Creates the view's private storage area as an actual subdirectory of *view-storage-dir-pname*. This subdirectory, named **.s**, holds checked-out versions, newly created derived objects, and other view-private objects.

–ln *remote-storage-dir-pname*

Creates the **.s** directory at another location, remote-storage-dir-pname. A UNIX-level symbolic link to *pname* is created at **view-storage-dir-pname/.s**, providing access to the remote storage area. Restrictions:

- **remote-storage-dir-pname** must be a valid pathname on every host (regardless of its network region) from which users will access the view.
- This view cannot be used to export a VOB to a non-ClearCase host. (See the **exports_ccase** reference page.)
- Some operations performed by **root** in this view may fail. This is another symptom of the **root**-becomes-nobody problem explained in *View Creator Identity and umask Permissions* on page 682.

This mechanism is independent of the network storage registry facility. The pathname to a remote storage area must be truly global, not global within a particular network region.

MARKING THE VIEW FOR EXPORT. *Default:* The view is not marked as an exporting view.

–nca·exported

Assigns an export-ID to the view-tag.

SETTING THE CACHE SIZE. *Default:* Set to the value of the site-wide default (set with **setcache –view –site**); if this default is not set, the cache size is set to 500 KB for a 32-bit platform and 1 MB for a 64-bit platform.

–cac·hesize *size*

Specifies a size for the **view_server** cache. *size* is an integer number of bytes, optionally followed by the letter **k** to specify kilobytes or **m** to specify megabytes; for example, **800k** or **3m**.

SPECIFYING THE KIND OF DERIVED OBJECTS TO CREATE IN A DYNAMIC VIEW. *Default:* **mkview** uses the site-wide default. If a site-wide default is not set, **mkview** configures the view to create shareable DOs.

–sha·reable_dos

Specifies that DOs created in the dynamic view can be winked in by other views.

–nsh·areable_dos
Specifies that DOs created in the dynamic view cannot be winked in by other views.

SETTING AN INITIAL DEFAULT FOR MODIFICATION TIMESTAMPS FOR A SNAPSHOT VIEW. *Default*: The initial default for the time stamps of files copied into the view as part of the snapshot view update operation is the time at which the file is copied into the view. Using the **update** command, users can change the default time-stamp mode: the most recently used time scheme is retained as part of the view's state and is used as the default behavior for the next update.

–pti·me
Changes the initial default for file time stamps copied into the snapshot view to the time at which the version was created (as recorded in the VOB).

ATTACHING A VIEW TO A STREAM. *Default:* None.

–str·eam *stream-selector*
Specifies a UCM stream. The view being created is attached to this stream.

stream-selector is of the form: [**stream:**]*stream-name*[*@vob-selector*] and *vob* is the stream's UCM project VOB.

SPECIFYING THE VIEW STORAGE DIRECTORY LOCATION. Either *dynamic-view-pname* or *snapshot-view-pname* is always a required argument. In addition, default behavior related to specifying view storage location is as follows:

Default for ClearCase and Attache dynamic views: None; a server storage location must be specified explicitly using **–stgloc** or indirectly using **–auto**.

For dynamic views, automatic server storage selection proceeds as follows:

1. Server storage locations that have no global path (**–ngpath**) are disqualified.
2. Server storage locations on heterogenous hosts are disqualified.
3. Local server storage locations are preferred over remote ones.
4. A server storage location is selected at random from the remaining candidates.

Default for ClearCase and Attache snapshot views: An automatically selected server storage location, if any can be found; else **–colocated_server**.

Default for ClearCase LT (snapshot) views: An automatically selected server storage location.

For snapshot views, automatic server storage selection proceeds as follows:

1. Server storage locations with global paths (**–gpath**) that reside on heterogeneous hosts are disqualified.
2. Local server storage locations are preferred over remote ones.
3. A server storage location is selected at random from the remaining candidates.

–stg·loc { *view-stgloc-name* | **–aut·o** }
Specifies a server storage location to hold the view storage directory (you must have previously used **mkstgloc** to create the server storage location). Either specify the server storage location by name, or specify **–auto** to indicate a server storage location is to be automatically selected as described previously.

When creating a snapshot view for disconnected use, use a server storage location; this way, the view can be moved without the need to use **register** to change registry information about the view's location.

You cannot create a view on a remote heterogeneous host unless the view is a snapshot views that is to be created in no-global-path (**–ngpath**) server storage location.

–col·ocated_server
Specifies that snapshot view's view storage directory (.view-stg) is to be created as a subdirectory of *snapshot-view-pname*. We recommend you use **–stgloc** rather than this option whenever possible.

–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*
See the **mkstgloc** reference page for information on these options.

NOTE: When you use one or more of the **–host**/**–hpath**/**–gpath** options in combination with **–colocated_server**, the values you specify for **–host**/**–hpath**/**–gpath** must correspond to *snapshot-view-pname*, not the colocated view storage directory.

dynamic-view-storage-pname
The location at which a new view storage directory is to be created for a dynamic view. (An error occurs if something already exists at this pathname.) You can create a view storage directory at any location in the file system where operating system permissions allow you to create a subdirectory, with these restrictions:

- You cannot create a view storage directory within a VOB, within another view, or within the root of the viewroot directory.
- *dynamic-view-storage-dir-pname* must specify a location on a host where ClearCase has been installed; the view database files must physically reside on a ClearCase host to enable access by the **view_server** process.

snapshot-view-pname
The location at which a new view storage directory is to be created for a snapshot view. (An error occurs if something already exists at this pathname.) You can create a view storage directory at any location in the file system where operating system permissions allow you to create a subdirectory, with these restrictions:

- You cannot create a view storage directory within a VOB, within another view, or within the root of the viewroot directory.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a dynamic view storage directory and assign it the view-tag **main_r2**.

cmd-context **mkview -tag mainr2 /net/host3/view_store/mainr2.vws**

```
Created view.
Host-local path: host3:/view_store/mainr2.vws
Global path: /net/host3/view_store/mainr2.vws
It has the following rights:
User : anne : rwx
Group: dev : rwx
Other: : r-x
```

- Create a dynamic view storage directory, assign it the view-tag **main_exp**, and mark it for export.

  cmd-context **mkview -tag main_exp -ncaexported /net/neon/views/main_exp.vws**

- Create a dynamic view storage directory named **Rel2.vws** in the current working directory, but with its private storage area on a remote host.

  cmd-context **mkview -tag Rel2 -ln /net/host4/priv_view_store/Rel2.vps Rel2.vws**

```
Created view.
Host-local path: host3:/view-store/Rel2.vws
Global path: /net/host3/view-store/Rel2.vws
It has the following rights:
User : anne : rwx
Group: dev  : rwx
Other:      : r-x
```

- Create a dynamic view on the local host. Then activate the view on a remote host.

cmd-context **mkview -tag anneRel2 /view_store/anneRel2.vws**

```
Created view.
Host-local path: host3:/view-store/anneRel2.vws
Global path: /net/host3/view-store/anneRel2.vws
It has the following rights:
User : anne : rwx
Group: dev : rwx
Other: : r-x
```

% rsh host4 cleartool startview anneRel2

The remote shell command is named **remsh** on some systems.

- Create a dynamic view storage directory, assign it the view-tag **smg_bigvw**, and specify a large cache size.

  cmd-context **mkview –tag smg_bigvw –cachesize 1m /home/smg/vws/smg_bigvw.vws**

  ```
  Created view.
  Host-local path: neon:/home/smg/vws/smg_bigvw.vws
  Global path:     /net/neon/home/smg/vws/smg_bigvw.vws
  It has the following rights:
  User : susan    : rwx
  Group: user     : rwx
  Other:           : r-x
  ```

- Create a snapshot view tagged **dev** with the view path **~bert/my_views**.

  cmd-context **mkview -tag dev -snapshot ~bert/my_views**

  ```
  Created view.
  Host-local path: peroxide:/export/home/bert/my_views/.view.stg
  Global path:     /net/peroxide/export/home/bert/my_views/.view.stg
  It has the following rights:
  User : bert : rwx
  Group: user     : r-x
  Other:          : r--
  Created snapshot view directory
  "/net/peroxide/export/home/bert/my_views".
  ```

- Create a UCM view and attach it to the specified stream.

  cmd-context **mkview -stream java_int@/vobs/core_projects -tag java_int /usr1/views/java_int.vws**

```
Created view.
Host-local path: propane:/usr1/views/java_int.vws
Global path:     /net/propane/usr1/views/java_int.vws
It has the following rights:
User : bill     : rwx
Group: user     : rwx
Other:          : r-x
Attached view to stream "java_int".
```

- Create a dynamic view at a server storage location that has been established for views.

  cmd-context **mkview -tag viewbert -stgloc view_stgloc**

```
Created view.
Host-local path: dioxin:/export/home/frank/view_stgloc/bert/viewbert.vws
Global path:
/net/dioxin/export/home/frank/view_stgloc/bert/viewbert.vws
It has the following rights:
User: bert : rwx
Group: user : rwx
Other: : r-x
```

SEE ALSO

chflevel, **chview**, **endview**, **lsview**, **mkstream**, **mkstgloc**, **mktag**, **registry_ccase**, **rmtag**, **rmview**, **setcache**, **setview**, **startview**, **umask**(1), **unregister**, **update**, **view**, **view_server**

mkvob

Creates and registers a versioned object base (VOB)

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only:

  mkvob –tag *vob-tag* [**–ucm**·**project**]
  [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
  [**–tco**·**mment** *tag-comment*] [**–reg**·**ion** *network-region*]
  [**–opt**·**ions** *mount-options*] [**–nca**·**exported**]
  [**–pub**·**lic**] [**–pas**·**sword** *tag-registry-password*]
  { [**–hos**·**t** *hostname* **–hpa**·**th** *host-storage-pname* **–gpa**·**th** *global-storage-pname*]
  vob-storage-pname
  | **-stgloc** { *vob-stgloc-name* | **–auto** } }

- ClearCase LT only:

  mkvob –tag *vob-tag* [**–ucm**·**project**]
  [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
  [**–tco**·**mment** *tag-comment*] [**-stg**·**loc** *vob-stgloc-name*]

NOTE: In ClearCase LT, you can run this command only on the ClearCase LT server host.

DESCRIPTION

The **mkvob** command creates a new versioned object base, or VOB, as follows:

- Creates a VOB storage directory at a specified path or in a VOB server storage location created with **mkstgloc**.
- Creates a VOB-tag with which the VOB is accessed by users.
- Places entries in the network's VOB registries; use the **lsvob** command to list registered VOBs.
- Starts a VOB server process on the host where the VOB storage directory physically resides.

A VOB storage directory is the root of a directory tree whose principal contents are a VOB database and a set of storage pools. See the **mkstgloc** and **vob** reference page for details.

VOB Directory Elements

mkvob creates the following directory elements in a VOB:

- **VOB root directory** — A **mkdir** command is implicitly executed to create a directory element—the VOB root directory—in the new VOB. Activating a VOB makes its root directory accessible at the pathname specified by the VOB-tag.
- **lost+found directory**—In ClearCase and Attache dynamic views only, **mkvob** also creates a special directory element, **lost+found**, as a subdirectory of the VOB root directory. In this directory are placed elements that are no longer entered in any versioned directory. **lost+found** is not accessible from a snapshot view.

See the **vob** reference page for more information on these directories.

Default Storage Pools

Each VOB storage directory is created with default storage pool subdirectories:

sdft	default source storage pool
cdft	default cleartext storage pool
ddft	default derived object storage pool (ClearCase and Attache dynamic views only)

Permissions

In considering access permissions, it is important to distinguish these two top-level directories:

- **VOB storage directory** — The standard directory created by this command, which is at the top level of a server storage location for VOBs.
- **VOB root directory** — The ClearCase or ClearCase LT directory element accessed at the VOB-tag.

When you create a VOB, your operating system-level UID and GID are assigned to the VOB storage and the default storage pools. The mode of the VOB storage directory is set according to your current **umask(1)** setting. This affects which users, and which views, can access the VOB. The modes of storage pool directories are set to 755, regardless of your current **umask** settng.

WARNING: Do not use operating system permission-setting utilities (for example, **chown(1)** or **chgrp(1)**) on a VOB storage directory. This creates inconsistencies and causes confusion.

The mode of the VOB root directory, by contrast, is derived from your current **umask(1)** setting. The mode can changed subsequently with the **protect** command. Note that the **w** permission on this directory (as on any directory element) affects only the creation of view-private objects;

changes to the VOB itself are controlled by ClearCase or ClearCase LT permissions, not those at the operating system level.

ClearCase, ClearCase LT, and Attache implement their own access scheme that goes beyond the standard operating system facilities. When you create a VOB, you become its VOB owner and your groups become its group list. These settings control access to many operations involving the VOB; they can be changed with the **protectvob** command.

See also the **protect** reference page (this command affects access to individual elements and shared derived objects) and *Administering ClearCase*.

Interop Text Mode Support

By default, VOBs are created with interop text mode support enabled. In this mode, the VOB database keeps track of the number of lines in all versions of each text file. This mode is required to support access to the VOB by interop text mode views (see the **mkview** reference page). To change the state of a VOB's interop text mode support, use the **msdostext_mode** utility. For more information, see *Administering ClearCase*.

ClearCase and Attache Only—Regional Tags

mkvob creates exactly one VOB-tag for the newly created VOB. This tag applies to the local host's network region by default. To make additional VOB-tags for other regions, use the **mktag** command. In general, the VOB-tags for a given VOB should all be public or all private.

ClearCase and Attache Dynamic Views Only—Public and Private VOBs

Some VOBs are to be shared, and others are to be used primarily by their creators. Accordingly, there are two kinds of VOB-tags: public and private.

PUBLIC VOB TAGS. A public VOB-tag specifies a location at which any dynamic-view user can mount the VOB. Furthermore, after a public VOB is mounted on a host, any user on that host can access it (subject to the standard access permissions).

Typically, all public VOBs are mounted at startup time with the command **cleartool mount –all**. (To create a public VOB that is not mounted automatically, specify **–options noauto** in the **mkvob** command.)

When creating a public VOB-tag with **mkvob** or **mktag**, you must supply the network's VOB-tag password; if you don't use the **–password** option, you are prompted to provide one. See the **registry_ccase** reference page for information on how the password is stored.

You need not create a public VOB's mount-over directory; the **cleartool mount** command creates it, if necessary.

PRIVATE VOB TAGS. A private VOB-tag specifies a mount point at which only the VOB's owner (usually, its creator) can mount the VOB using **cleartool**. For example:

cleartool mount /vobs/myPrivateVob

root can use the **cleartool mount** *vob-tag* command to bypass the "owner only" mount restriction. The command **cleartool mount –all** does not mount private VOBs.

After a private VOB is mounted, any user can access it (subject to the standard access permissions). You must explicitly create the mount-over directory for a private VOB; the **cleartool mount** command does not create it automatically.

PRIVATE-TO-PUBLIC CONVERSION. To convert a private VOB to a public VOB, use a command like this:

cmd-context **mktag -vob -tag /vobs/vob3.p -replace -public /usr/vobstore/private3.vbs**

This replaces the VOB's private VOB-tag with a public one. **mktag** prompts you to enter the VOB-tag password.

ClearCase, Attache, and ClearCase LT Snapshot Views Only—Accessing Public and Private VOBs

For an explanation of public and private VOBs, see *ClearCase and Attache Dynamic Views Only—Activating the VOB* on page 693.

- ClearCase and Attache—Snapshot views make no distinction between public and private VOBs: you can access private VOBs from a snapshot view regardless of who owns them.
- ClearCase LT—All VOBs are private and can be accessed from any view.

ClearCase and Attache Dynamic Views Only—Activating the VOB

A VOB cannot be used for development work in a dynamic view until it is activated with the **cleartool** or Attache **mount** command. This causes the VOB's storage directory to be mounted on the host at the VOB-tag location, as a file system of type MVFS. See the **mount** reference page for details.

ClearCase Only—Marking a VOB for Export

A VOB to be used by some view for NFS access (non-ClearCase access) must be marked for export. Each export VOB is assigned an export-ID, which ensures that NFS-exported view/VOB combinations have stable NFS file handles across server reboots or shutdown and restart of ClearCase.

If the VOB is registered in multiple regions, the export marking must appear on all of that VOB's tags in all the regions in which it is registered. To mark a VOB for export, use the **–ncaexported** option. To mark an existing VOB for export, use **mktag –replace –ncaexported**.

The VOB export-ID is stored in the mount options field in the VOB-tag registry. If you use the **–ncaexported** option and specify additional mount options in the **mktag** or **mkvob** command, the mount options field includes an appropriate export-ID mount option.

For information on exporting VOBs, see the **export_mvfs** reference page.

NOTE: Marking a VOB for export is not required for NFS export to work, but it is required if you want to avoid "stale file handle" problems after a server restart.

CAUTION: After you create a VOB, do not move the VOB database directory (**db**) to another host. The VOB database directory must be on the host where the VOB storage directory physically resides. Moving the VOB database directory can negatively affect both VOB performance and ClearCase's ability to ensure the integrity of the VOB database by recovering from interrupted transactions.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE VOB-TAG. *Default:* None.

–tag *vob-tag*
VOB tags are names for VOBs that are entered in the registry and are of either single-component (**/vob1**) or multicomponent (**/vob/src**) form. The VOB tag is where the VOB appears under the view root.

ClearCase and Attache dynamic views only—A pathname—typically multicomponent—that specifies the mount-over directory at which the VOB is mounted as a file system of type MVFS . The VOB-tag is entered in the VOB tag registry. If you are creating a private VOB (no **–public** option), you must also create the mount-over directory on each host where you will mount the VOB. (The **cleartool mount** command creates mount-over directories for public VOBs.)

ClearCase LT only—The VOB tag must be of the single-component form.

ClearCase and Attache only—If your network has multiple regions from which the VOB is to be accessed, use **mktag** to create an additional VOB-tag for each region.

SPECIFYING THE KIND OF VOB. *Default:* A standard (that is, nonproject) VOB.

–ucm·project
Creates a UCM project VOB for storing UCM-related objects including activities, baselines, components, folders, projects, and streams. Typically, a single project VOB is shared by multiple source VOBs—those that store versioned source code, documents, and so on.

ClearCase LT only—You cannot create more than one project VOB.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by your **.clearcase_profile** file (default: **–cqe**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

–tco·mment *tag-comment*
Adds a comment to the VOB-tag's entry in the **vob_tag** registry file. Use **lsvob –long** to display the tag comment.

SPECIFYING A NETWORK REGION. *Default:* Creates the VOB-tag in the local host's network region. (Use the **hostinfo –long** command to display the network region.) See the **registry_ccase** reference page for a discussion of network regions.

–reg·ion *network-region*
Creates the VOB-tag in the specified network region. An error occurs if the region does not already exist.

CAUTION: The VOB-tag created with **mkvob** must be for the network region to which the VOB host belongs. Thus, use this option only when you are logged in to a remote host that is in another region. Moreover, a VOB-tag for the VOB's home region must always exist.

SPECIFYING MOUNT OPTIONS. *Default:* Mounts each VOB using the **–options** field in its **vob_tag** registry file.

–opt·ions *mount-options*
Options to be used in mounting the VOB. The following options are valid: **ro**, **rw**, **soft**, **hard**, **intr**, **nointr**, **noac**, **noauto**, **nodev**, **nodnlc**, **nosuid**, **suid**, **retrans**, **timeo**, **acdirmin**, **acdirmax**, **acregmin**, **acregmax**, **actimeo**

See the appropriate operating system reference page (for example, **mount(1M)**) for the meanings of these options. If the mount options list contains white space, enclose it in quotes.

By default, a VOB is mounted in **nointr** mode. This means that operations on MVFS files (for example, **open(2)**) cannot be interrupted by typing the **INTR** character (typically, CTRL+C). To enable keyboard interrupts of such operations, use the **intr** mount option.

MARKING THE VOB FOR EXPORT. *Default:* The VOB is not marked for export.

–nca·exported
Assigns an export-ID to the VOB. See *ClearCase Only—Marking a VOB for Export* on page 693.

PUBLIC VS. PRIVATE VOB. *Default:* A private VOB.

–pub·lic
Creates a public VOB. See *ClearCase and Attache Dynamic Views Only—Activating the VOB* on page 693.

–pas·sword *tag-registry-password*

A password is required to create a public tag or to create a private tag when you include **suid** as an argument to **–options**.

In these cases, if you do not include the VOB-tag password, **mkvob** prompts for it. See the **registry_ccase** reference page. An error occurs if there is no match. Note that the VOB is created, but without a VOB-tag. Use **mktag** to supply a public or private VOB-tag.

CAUTION: This is a potential security breach, because the password remains visible on the screen.

SPECIFYING THE VOB'S LOCATION AND NETWORK ACCESSIBILITY. *Default for ClearCase and Attache:* None. *Default for ClearCase LT:* The server storage location on the ClearCase LT server host with the most free space.

–hos·t *hostname*
–hpa·th *host-storage-pname*
–gpa·th *global-storage-pname*

See the **mkstgloc** reference page for information on these options.

vob-storage-pname

The location at which a new VOB storage directory is to be created: full pathname, relative pathname, or simple subdirectory name. (An error occurs if something already exists at this pathname.) You can create a VOB at any location where the operating system allows you to create a subdirectory, with these restrictions:

- You cannot create a VOB within an existing VOB storage directory.
- You cannot create a VOB under an existing VOB-tag (VOB mount point).
- You cannot create a VOB within the viewroot directory (**/view**).
- *vob-storage-pname* must specify a location on a host where ClearCase or has been installed. The VOB database (located in subdirectory **db** of the VOB storage directory) must physically reside on a ClearCase host, where it is accessed by ClearCase server programs running locally.

–stg·loc { *vob-stgloc-name* | **–aut·o** }

Specifies a server storage location in which the VOB storage directory is to be created. The server storage location must have been created previously with **mkstgloc**. You can specify the name of the VOB server storage location explicitly as *vob-stgloc-name*, or specify **–auto** to direct **mkvob** to select one.

If you specify **–auto**, a server storage location for the VOB is selected as follows:

a. Server storage locations that have no global path (**mkstgloc –ngpath**) and that reside on remote hosts are disqualified.

b. Server storage locations on heterogenous hosts are disqualified.

c. Local server storage locations are preferred over remote ones.

d. Globally accessible server storage locations (**mkstgloc –gpath**) are preferred over those that are not (**mkstgloc –ngpath**).

e. The server storage location with the most free space is selected.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Create a private VOB storage directory, **project3.vbs**, in the **/usr/vobstore** directory on local host **venus**, and give it the VOB-tag **/vobs/project3**. Then, mount the VOB on the local host.

```
cmd-context  mkvob -tag /vobs/project3 -c "main development sources" \
/usr/vobstore/project3.vbs
Created versioned object base.
Host-local path: venus:/usr/vobstore/project3.vbs
Global path: /net/venus/usr/vobstore/project3.vbs
VOB ownership:
 owner anne
 group dev
Additional groups:
 group usr
 group adm
```

```
% mkdir /vobs/project3                    (create VOB mount point to match the VOB-tag)
cmd-context  mount /vobs/project3         (mount VOB as file system of type MVFS)
```

- Create a public VOB, which will be mounted at startup time (by all hosts in the current host's network region), and mark it for export.

cmd-context **mkvob -tag /vobs/src1 -public -password tagPword **
-ncaexported /vobstore/src1.vbs

```
Created versioned object base.
Host-local path: saturn:/vobstore/src1.vbs
Global path: /net/saturn/vobstore/src1.vbs
.
.
.
```

- Create a private VOB in a different region, explicitly specifying the registry information.

  cmd-context **mkvob –tag /vobs/doctools –c "storage for documentation tools" **
  **–region unix_dev –host neon –hpath /vobstg/doctools.vbs **
  –gpath /net/neon/vobstg/doctools.vbs /vobstg/doctools.vbs

```
Created versioned object base.
Host-local path: neon:/vobstg/doctools.vbs
Global path: /net/neon/vobstg/doctools.vbs
.
.
.
```

- Create a VOB at VOB server storage location.

  cmd-context **mkvob -tag /vobbert -stgloc stgloc1**

```
Comments for "/export/home/bert/stgloc1/vobbert.vbs":
test vob
.
Created versioned object base.
Host-local path: peroxide:/export/home/bert/stgloc1/vobbert.vbs
Global path:     <no-gpath>
cleartool: Warning: This global path value precludes use of this VOB by
dynamic views from region "test_region".
.
.
.
```

SEE ALSO

chpool, **lsvob**, **mkpool**, **mkstgloc**, **mount**, **protectvob**, **registry_ccase**, **rgy_passwd**, **rmvob**, **uncheckout**, **umount**, **vob**, **umask(1)**

mkws

Makes a *workspace* associated with a *dynamic view*

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

- Make and register a workspace and associate it with an existing dynamic view:
  mkws [**–sho·st** *hostname*] **–tag** *tagname ws-stg-pname*
- Make and register a workspace and a create a new associated dynamic view:
  mkws –hos·t *hostname* **–hpa·th** *host-stg-pname* **–gpa·th** *global-stg-pname*
  [**–sho·st** *hostname*] **–tag** *tagname ws-stg-pname*

DESCRIPTION

The **mkws** command creates and adds a workspace to the local *workspace registry* and either associates it with an existing dynamic view or creates a new associated dynamic view. *ws-stg-pname* specifies the location of the *workspace storage directory. tagname* specifies the *workspace name* which is also the associated view's tag. A username and password combination for the *workspace helper host* are required. You are prompted for this information if it has not already been requested, or previously stored using the **Login info** command on the **Options** menu. After the workspace is created, it becomes the current workspace.

Attache's Client Process Startup Directory

There is a separate startup directory associated with the Attache client process. This directory changes depending upon how Attache is started. For example, it is the "working directory" specified in Attache's program item properties if Attache is started from the icon. Once the Attache client process is started, this directory never changes. The pathname of a new *workspace storage directory* (if not specified absolutely) is relative to the Attache startup directory, not your workspace *working directory*. For this reason, we recommend that you always specify a *full local pathname* for your workspace storage directory.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE ATTACHE HELPER'S LOCATION. *Default*: View host.

–sho·st *hostname*
Specifies the name of the ClearCase host on which the associated Attache helper process will run.

SPECIFYING THE WORKSPACE NAME. *Default*: None.

–tag *tagname*
Specifies the name of the workspace and dynamic view, in the form of a simple *helper host* file name. *tagname* specifies both the *view-tag* name and the *workspace name*, which are created if they do not exist.

SPECIFYING THE WORKSPACE STORAGE DIRECTORY. *Default*: None. You must specify a location for the *workspace storage directory.*

ws-stg-pname
Specifies the name of the workspace storage directory: full pathname, relative pathname, or simple directory name. This directory can already exist, but if it doesn't, it is created. As with any operating-system directory-creation command, the entire directory tree above the workspace storage directory name must already exist. A relative pathname or simple directory name begins from Attache's startup directory, not the working directory.

SPECIFYING THE NEW DYNAMIC VIEW'S LOCATION. *Default*: None..

–hos·t *hostname*
–hpa·th *host-stg-pname*
–gpa·th *global-stg-pname*
See the **mkstgloc** reference page for descriptions of how to use these options and arguments to specify VOB and view storage directories.

Values of other view creation options (**–tcomment**, **–tmode**, **–ln**, **–region**) are provided by default. To control these attributes of view creation, use **mkview** instead and then use **mkws** to connect to this dynamic view. The default behavior for text mode can also be specified with the **Preferences** command on the **Options** menu.

EXAMPLES

- Make a workspace and associate it with the existing dynamic view **jo_doc**. At an Attache prompt:

  mkws –shost darkover –tag jo_doc c:\users\jo\doc

- Make a workspace and create a new dynamic view named **lee_main** to associate with it. This command must be entered on a single line. At an Attache prompt:

  mkws –host oz –hpath /usr /lee/vws/mn.vws –gpath /net/oz/usr/lee/vws/mn.vws –tag lee_main c:\users\lee\main

SEE ALSO

attache_graphical_interface, attache_command_line_interface, lsview, lsws, mkview, rmws, setws

mount

Activates a VOB at its *VOB-tag* directory

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

- Mount a single VOB:

  mount [**–opt·ions** *mount-options*] *vob-tag*

- Mount all public VOBs:

  mount –a·ll

DESCRIPTION

Prerequisite: The VOB being activated must already have a VOB-tag for your host's network region in the ClearCase registry. See the **mkvob** and **mktag** reference pages.

The **mount** command activates one or more VOBs on the local host by performing UNIX-level mounts of their VOB storage directories. The **mount** command invokes a short-lived **mntrpc_server** process on the VOB host; this process, running as **root**, performs the mount. The **mount** command mounts a VOB as a file system of type MVFS (multiversion file system) and is inapplicable to non-MVFS installations.

Mounting All VOBs

The **mount –all** command mounts all public VOBs listed for your host's network region in the VOB registry. This command executes at ClearCase startup time; see the **init_ccase** reference page. (It does not mount private VOBs or VOBs whose tag entries include the mount option **noauto**.)

Mounting of Public and Private VOBs

A public VOB can be activated by any user; if the mount-over directory does not already exist, it is created.

A private VOB can be activated only by its owner. The *root* user or VOB owner can use the standard **mount(1M)** command to mount a private VOB; other users cannot mount it. The mount-over directory must already exist and be owned by the VOB owner.

Only the *root* user can use **–options** to specify mount options on the command line. See the **permissions** reference page.

See the **mkvob** reference page for a discussion of *public* and *private* VOBs.

VOB-TAGS AND THE VOB STORAGE REGISTRY

You reference a VOB by its *VOB-tag* (the full pathname of its mount point), not by its storage area pathname. The **mount** command uses the VOB-tag to retrieve all necessary information from the *ClearCase registry*: pathname of VOB storage area, pathname of mount point, and mount options.

PERMISSIONS AND LOCKS

Permissions Checking: See *Mounting of Public and Private VOBs*. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING MOUNT OPTIONS. *Default:* Mounts each VOB using the **–options** field in its *vob_tag* registry file.

–opt·ions *mount-options*
(*root* user only; mutually exclusive with **–all**) Ignores the **–options** field in the *vob_tag* registry file entry and uses the specified set of options, which can include these:

nodev, **nosuid**, **suid**, **ro**, **rw**, **soft**, **hard**, **intr**, **nointr**, **timeo**, **retrans**, **noauto**, **nodnlc**, **noac**, **acdirmin**, **acdirmax**, **acregmin**, **acregmax**, **actimeo**

See the appropriate operating system reference page (for example, **mount(1M))** for a description of these options. Enclose this argument in quotes if it contains white space.

If you don't specify a time-out or retransmission option, default values are used:

timeo=5 (seconds)
retrans=7 (retries)

By default, a VOB is mounted in **nointr** mode. This means that operations on MVFS files (for example, **open(2)**) cannot be interrupted by typing the **INTR** character (typically, CTRL+C). To enable keyboard interrupts of such operations, use the **intr** mount option.

SPECIFYING THE VOB(S). *Default:* None.

vob-tag
Mounts the VOB with this *VOB-tag*, which must be specified exactly as it appears in the **vob_tag** registry file. Use **lsvob** to list VOBs.

–a·ll
(Mutually exclusive with **–options**) Mounts all public VOBs listed for your host's network region in the VOB registry, using the mount options in their *vob_tag* registry entries. (Including the mount option **noauto** in a *VOB-tag*'s registry entry prevents the VOB from being mounted by **mount –all**.)

IMPLEMENTATION NOTE

mount calls the standard **mount(1M)** command, which in turn calls the ClearCase utility **mount_mvfs** to perform the actual work.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Mount the VOB storage directory that is registered with *VOB-tag* **/vobs/Rel4**.

  cmd-context **mount /vobs/Rel4**

- Mount all VOBs registered with public *VOB-tag*s.

  cmd-context **mount –all**

SEE ALSO

lsvob, **mkvob**, **mktag**, **mount_ccase**, **register**, **registry_ccase**, **umount**, **mount(1M)**

mount_ccase

Mount/unmount commands for VOBs and the viewroot directory

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

The **mount_mvfs** program must never be invoked explicitly.

The **cleartool mount** subcommand invokes an architecture-specific mount command:

Solaris, Reliant UNIX	**/usr/lib/fs/mvfs/mount**
AIX 4	**/sbin/helpers/mvfsmnthelp**
Digital UNIX	**/sbin/mount_mvfs**
IRIX, MP-RAS	**/usr/etc/mount_mvfs**
HP-UX 10, HP-UX 11	**/sbin/fs/mvfs/mount**
UnixWare	**/etc/fs/mvfs/mount**

DESCRIPTION

This reference page describes the mechanisms that mount VOBs as file systems of type MVFS (the ClearCase *multiversion file system*).

AUTOMATIC VOB ACTIVATION AT SYSTEM STARTUP. At system startup, the architecture-specific ClearCase startup script (see the **init_ccase** reference page) issues a **mount –all** command. This activates on the local host all the VOBs that are registered as *public* in the (local host's *network region* of the) ClearCase *tags registry*. During this procedure, the architecture-specific mount command performs the actual work of mounting the VOB as a file system of type MVFS. (The command is actually a symbolic link to *ccase-home-dir***/etc/mount_mvfs**.)

VOB ACTIVATION AFTER SYSTEM STARTUP. After system startup, a **mount** command can be used to activate or reactivate any VOB that is listed in the tags registry.

- The *root* user can activate any VOB in this way.
- A nonroot user can activate any public VOB, or any private VOB owned by that user.

AUTOMATIC VOB DEACTIVATION AT SYSTEM SHUTDOWN. At system shutdown, the architecture-specific ClearCase startup script is invoked with the **stop** option to execute the ClearCase shutdown procedure. As part of this procedure, a **umount –all** command deactivates all VOBs currently active on the local host. On all platforms except for AIX 4, **umount –all**

invokes the standard **umount(1M)** utility directly. On AIX 4, **umount –all** invokes the architecture-specific mount command **/sbin/helpers/mvfsmnthelp** with **U** as its first argument, and **/sbin/helpers/mvfsmnthelp** then invokes **umount(1M)**.

INDIVIDUAL VOB DEACTIVATION. While ClearCase is running, a **umount** command can be used to deactivate any mounted VOB:

- The *root* user can deactivate any VOB in this way.
- A nonroot user can deactivate any *public* VOB, or any *private* VOB owned by that user.

SEE ALSO

exports_ccase, **mount**, **umount**, **mount(1M)**, **umount(1M)**, **mountall(1M)** [some architectures]

msdostext_mode

Enables or disables a VOB's interop text mode support

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

ccase-home-dir/**etc/utils/msdostext_mode** [**–c** | **–d**] *vob-stg-pname*

DESCRIPTION

Before a VOB can be accessed from an interop text-mode view, the VOB must be enabled for interop text mode support. The **msdostext_mode** utility enables or disables the ability of a VOB to support interop text mode views. This utility does not physically convert or modify files in any way; rather, it affects the information recorded for text file versions in the VOB database.

To enable a replicated VOB for access by MS-DOS text-mode views:

1. Synchronize all replicas in the VOB family.
2. At each site, lock the VOB using the following command:

   cleartool lock –nuser root-*or-vob-owner* **vob:***pname-in-vob*
3. As **root** or VOB owner, run **msdostext_mode** on all replica storage directories.
4. Unlock all replicas.

For a detailed discussion of interop text mode, see *Administering ClearCase*.

PERMISSIONS AND LOCKS

Permissions Checking: You must be one of the following: VOB owner, **root**. See the **permissions** reference page.

Locks: The command fails if the VOB is locked for the user running **msdostext_mode**.

OPTIONS AND ARGUMENTS

With no options, **msdostext_mode** does the following:

1. Directs the VOB to store line counts in the VOB database for all versions of all elements of type *text_file* and *compressed_text_file* (and any element types derived from these).

2. Enables interop text mode support, so that line counts can be recorded for newly created versions.

–c

Converts the VOB but does not enable interop text mode support. Running **msdostext_mode** periodically (as a **cron** job, for example) with **–c** enabled offers a small performance advantage over having the VOB continually track file sizes for all new versions. The disadvantage is that recorded file sizes become increasingly inaccurate as new versions are checked in between invocations of **msdostext_mode –c** For this reason, we do not recommend this usage of the utility..

Do not use this option for the initial conversion of a VOB. This option is intended to allow for conversions of a replicated VOB subsequent to its initial conversion so that any elements replayed from a VOB that is not enabled for interop text mode support can get line counts.

–d

Disables interop text mode support.

vob-stg-pname

Storage directory pathname of the VOB.

SEE ALSO

mkview, *Administering ClearCase*

mv

Moves or renames an element or VOB link

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- Rename:

  mv | **move** [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**] *pname target-pname*

- Move to another directory:

  mv | **move** [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**] *pname* [*pname ...*] *target-dir-pname*

DESCRIPTION

NOTE: The directory where the element to be moved or renamed resides must be checked out. The destination directory must also be checked out; this directory may be the same as the source directory. **mv** appends an appropriate line to the checkout comment for all relevant directories.

The **mv** command changes the name or location of an *element* or *VOB symbolic link*. For a file element that is checked out to your view, it relocates the checked-out version, also. (That is, it moves the view-private file with the same name as the element.) If the version is checked out to another view, it issues a warning:

```
cleartool: Warning: Moved element with checkouts to "overview.doc";
     view private data may need to be moved.
```

The **mv** command can move an element only within the same VOB. To move an element to another VOB, use the **relocate** command.

NOTE: The **mv** command does not affect the previous versions of the directory containing the element. If you set your config spec to select a previous version of the directory, you see the old name of the element.

Moving in Attache

In Attache, if the move operation would overwrite an existing writable file or directory subtree containing writable files in the workspace, a confirming query is issued. Otherwise, local files or directories corresponding to successfully renamed elements in the view are moved or renamed as well.

Moving in Snapshot Views

When you move a file element in a *snapshot view*, only the to/from pathnames you specify are updated in the view. If the view contains multiple copies of the element (because VOB symbolic links or hard links exist), the copies are not updated. To update the copies, you must use the **update** command.

If the move operation would overwrite a writable file or directory subtree containing writable files, **mv** renames the files to *filename***.renamed**.

Moving View-Private or Attache Workspace Objects

This command is for VOB-database objects. To rename or move view-private files, use the standard **mv(1)** command. To rename or move local files in the Attache workspace, use the Windows **rename** or **move** command in a DOS window or in the File Manager.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. An error occurs if the VOB is locked.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE EXISTING OBJECTS. *Default:* None.

pname
One or more pathnames, specifying elements or VOB links. If you specify more than one *pname*, you must specify a directory (*target-pname*) as the new location.

SPECIFYING THE NEW LOCATION. *Default:* None.

target-pname
The new location for the single element or VOB link specified by *pname*. Both *pname* and *target-pname* must specify locations in the same VOB. An error occurs if an object already exists at *target-pname*.

target-dir-pname
The pathname of an existing directory element, to which the elements or links are to be moved. This directory must be located in the same VOB as the objects being moved.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In all the examples, all directories involved must be checked out.

- Rename a C-language source file from **hello.c** to **hello_1.c**.

  cmd-context **mv hello.c hello_1.c**
  ```
  Moved "hello.c" to "hello_1.c".
  ```

- Move all files with a **.c** extension into the **src** directory.

  cmd-context **mv *.c src**
  ```
  Moved "cm_add.c" to "src/cm_add.c".
  Moved "cm_fill.c" to "src/cm_fill.c".
  Moved "convolution.c" to "src/convolution.c".
  Moved "hello.c" to "src/hello.c".
  Moved "hello_old.c" to "src/hello_old.c".
  Moved "messages.c" to "src/messages.c".
  Moved "msg.c" to "src/msg.c".
  Moved "util.c" to "src/util.c".
  ```

- Rename a symlink from **messages.c** to **msg.lnk**, and show the result with **ls**.

  cmd-context **mv messages.c msg.lnk**
  ```
  Moved "messages.c" to "msg.lnk".
  ```

  cmd-context **ls -long msg.lnk**
  ```
  symbolic link msg.lnk --> msg.c
  ```

SEE ALSO

checkout, **cd**, **ln**, **ls**, **relocate**, **update**

mvfscache

Controls and monitors MVFS caches (*dynamic view* only)

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

- Determine cache status:

  ccase-home-dir/**etc/mvfscache** [*cache_name*]

- Control cache operation:

  ccase-home-dir/**etc/mvfscache** { **–e** *cache_list* | **–d** *cache_list* | **–f** *cache_list* }

- Display name cache contents:

  ccase-home-dir/**etc/mvfscache –p** [**–n** *name*] [**–v** *dbid*] [**–i**]

DESCRIPTION

NOTE: This utility is not intended for general use. It is intended primarily to help ClearCase engineering and Rational Technical Support personnel diagnose problems with the MVFS. For information on user-level cache commands, see the **getcache** and **setcache** reference pages.

mvfscache manipulates a dynamic view host's *MVFS caches*, which are used to optimize file system performance. The root user can enable or disable a cache. Any user can flush a cache.

OPTIONS AND ARGUMENTS

DETERMINING CACHE STATUS. With no options or arguments, **mvfscache** displays the enabled/disabled status of all MVFS caches. If you don't use any of the options, but specify a cache name as an argument, **mvfscache** does not display any output; it returns an appropriate exit status:

0	specified cache is enabled
1	specified cache is disabled

CONTROLLING CACHE OPERATION. Use one of the following options to control a cache, a set of caches, or cache-related behavior.

–e *cache-list*
(Must be root) Enables the specified caches and (with **cto**) cache-related behavior. Descriptions of these caching parameters are in *Administering ClearCase*. The *cache-list*

must be comma-separated, with no white space, and can include one or more of the following keywords.

attr Attribute cache. Caches **stat(2)** records for recently accessed file-system objects.

name Name cache. Caches name lookup translations for recently accessed files and directories.

slink Symbolic link text cache. Caches the contents of recently accessed symbolic links.

rvc VOB root version cache. Caches VOB mount point data for each dynamic view.

cto (cache-related behavior) Close-to-open consistency behavior. Forces a **stat** operation to the **view_server** on every operating-system open operation.

Disabling this behavior may boost performance if **mvfsstat** or **mvfstime** shows heavy **cto** activity and the user is not sharing views. However, disabling this behavior may result in consistency loss.

–d *cache-list*

(Must be root) Disables the specified caches and cache-related behavior. The syntax is the same as for **–e**.

–f *cache-list*

Flushes the specified caches. Use this option only under direction from Rational Technical Support. The *cache-list* can include any number of the following keywords; the list must be comma-separated, with no white space.

mnode Mnode freelist cache. Flushes the `attr` and `slink` caches, open freelist files, and mnode storage for all freelist mnodes.

name Name cache.

rvc VOB root version cache.

lcred Global credentials cache for cleartext lookup permissions.

DISPLAYING NAME CACHE CONTENTS. Use **–p** by itself or with one or more of **–n**, **–v**, and **–i**. The name cache contains the name lookup translations for recently accessed files and directories. The first line of a name lookup translation has this form:

VOB-tag *view*:*directory-dbid* *name* ==> *view*:*lookup-dbid*

–p

Prints the contents of the name cache.

–n *name*

Prints only the entries in the name cache that match *name*.

–v *dbid*

Prints only the entries in the name cache that match *directory-dbid* (database-ID for the directory in which *name* is found) or *lookup-dbid* (database-ID for the result of the lookup).

–i

Includes invalidated name cache entries in the output. These are entries that have been marked bad and are not used in lookups, but are retained for statistical purposes. This helps determine how often invalid entries are replaced with new data. Invalidations usually happen when **cleartool** or **clearmake** changes something in the VOB and knows that the MVFS needs to refetch that information for its cache.

EXAMPLES

- Determine the status of all caches.

  /usr/atria/etc/mvfscache

  ```
  Attr: on
  Name: on
  Rvc: on
  Slink: on
  Cto: on
  ```

- Clear busy mount points, to prepare for unmounting VOBs.

  /usr/atria/etc/mvfscache –f mnode

- Enable the **name** and **attr** caches:

  /usr/atria/etc/mvfscache –e name,attr

SEE ALSO

mvfslog, **mvfsstat**, **mvfsstorage**, **mvfstime**, **mvfsversion**, **csh(1)**, **stat(2)**

mvfslog

Sets or displays MVFS console error logging level

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

ccase-home-dir/**etc/mvfslog** [**–kern**·**log** *file* | **–nokern**·**log**]
[**none** | **error** | **warn** | **info** | **stale** | **debug**]

DESCRIPTION

The **mvfslog** command sets or displays the verbosity level and location for MVFS console error logging. The initial setting is **error**, in which only RPC errors and actual MVFS errors are logged; warnings and diagnostics are suppressed.

Each logging level includes messages from the previous levels. For example, **info** includes messages from **error** and **warn**.

PERMISSIONS AND LOCKS

Permissions: Any user can display the logging level and location. To change the level or location, you must be the root user.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING A LOG FILE. *Default:* MVFS output is written to **/var/adm/atria/log/mvfs_log**.

–kern·**log** *file*
Specifies a log file for MVFS output.

–nokern·**log**
Closes the log file and directs MVFS output to the system console.

SETTING THE LOGGING LEVEL. *Default:* Displays the current error logging level. Use one of the following keywords to specify a new level; **none** is the least verbose; **debug** is the most verbose.

none	RPC errors only.
error	MVFS errors are logged (default setting).
warn	MVFS warnings are logged.
info	MVFS diagnostics on some expected errors are logged.
stale	MVFS diagnostics related to ESTALE errors are logged.

debug Verbose information on many expected errors.

SEE ALSO

mvfscache, **mvfsstat**, **mvfsstorage**, **mvfstime**, **mvfsversion**

mvfsstat

Displays MVFS statistics

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

ccase-home-dir/**etc/mvfsstat** [**–icrvVhalzAd**] [**–o** *outfile*] [*time*] [*count*]

DESCRIPTION

NOTE: This utility is not intended for general use. It is intended primarily to help ClearCase engineering and Rational Technical Support personnel diagnose problems with the MVFS. For information on improving ClearCase client performance, see *Administering ClearCase*.

The **mvfsstat** command displays MVFS usage and operating statistics, including cumulative statistics on MVFS cache usage, **rpc** statistics, cleartext I/O counts, **vnode** operation counts, and **VFS** operation counts. This data is useful for evaluating file-system performance and determining whether MVFS cache sizes require adjustment.

MVFS CACHE STATISTICS

The **–c** option reports on the usage of the host's MVFS caches. This report is cumulative, covering the entire period since the MVFS was reloaded. The following example covers a 23-day period:

```
----------------- Fri Jul 16 16:20:16 1999 --------------------
dnlc:   2267082  2229727( 98.4%) hit    1301984 dot 846301 dir 62901 reg 18541 noent
                   37355(  1.6%) miss     25099 events
                   42761(  1.9%) add      27820 dir 7368 reg 7573 noent
attr:   2355186  2349946( 99.8%) hit        120 lvut
                    5240(  0.2%) miss (3902cto+0gen+970timo+74new,294ev;35lvut)
                     57302 updates      5546unexp+2206exp mod, 2295 vmod
```

The following sections describe the particular statistics that are useful in tuning MVFS performance on a ClearCase client host.

Directory Name Lookup Cache (dnlc)

The `dnlc` section reports on usage of a name-lookup cache that maps pathnames to ClearCase identifiers. Note that the value precedes the keyword. For example, `1301984 dot` means that the reported value of the "dot" statistic is 1301984.

Cache Hits. The `hit` line reports on the number of times an entry type was found in the cache (hit):

`dot`	Number of times the current working directory was looked up (always a cache hit)
`dir`	Number of times a directory object entry was found in the cache.
`reg`	Number of times a file object entry was found in the cache.
`noent`	Number of times a cached `File not found` (`ENOENT`) entry was found.

This cache has low hit rates (around 50%) for activities that walk a large tree—for example, a **find** command, or a recursive **clearmake** that examines many files and determines that nothing needs to be built.

Cache Misses. The `miss` line reports on total cache misses. The `events` value is the number of cache misses that occurred because of a significant VOB event, a time-out of the entry, or vnode recycling. Cache misses can occur because there was no entry in the cache. The total number of cache misses equals the `events` value plus the number of misses occurring because there was no entry in the cache.

Cache Additions. The `add` line reports on cache misses that occurred because a new entry was being added to the cache. The additions are categorized as directory entries (`dir`), file entries (`reg`), and ENOENT entries (`noent`).

Attribute Cache

The `attr` section reports on usage of a cache of **stat(2)** returns.This cache generally has hit rates comparable to that for the directory name lookup cache.

OPTIONS AND ARGUMENTS

time
: Time in seconds between samples. Display deltas on each sample. If you omit this option, only the absolute values of all information are printed.

count
: Number of samples. If omitted, defaults to “infinite”.

–o *outfile*
: Writes the output to *outfile*.

–i
: Display cleartext I/O counts and wait times.

–c
: Display statistics for the MVFS caches, as described in *MVFS CACHE STATISTICS* above.

–r
: Display MVFS remote-procedure-call (RPC) statistics. These statistics include both

counts and real-time waited. Real-time waited may be greater than 100% of a sample period in two cases:

- When an operation took longer to complete than the sample period; for example, 60 seconds of wait time is recorded in a 30-second sample.
- Multiple processes are waiting at the same time.

In general, real-time percentages are meaningful only when a single process is accessing a VOB.

–v

Displayscounts of **vnode** operations.

–V

Displays counts of **vfs** operations.

–h

Displays an RPC histogram. Cleartext fetch RPCs are tallied separately from all other RPCs.

–a

Displays auditing statistics.

–l

Adds more detail to the statistics generated by **–c**, **–r**, **–i**, **–v**, and/or **–V**, by providing a breakdown by individual operations. With **–c**, also provides per-cache-entry hit ratios.

–z

Must be *root user*. Resets all running counters to zero.

–A

Displays all statistics.

–d

With **–c** or **–A**, displays additional debugging information for use in diagnosing problems. Use this option only under direction from Rational Technical Support.

SEE ALSO

mvfscache, **mvfslog**, **mvfsstorage**, **mvfstime**, **mvfsversion**

mvfsstorage

Lists data container pathname for MVFS file

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

ccase-home-dir/**etc/mvfsstorage** *pname ...*

DESCRIPTION

mvfsstorage lists the pathname of an MVFS file's data container. The pathname may be within view-private storage, the source pool, or the cleartext pool, depending on the kind of file.

For a ...	mvfsstorage displays
View-private file (including checked-out versions, unshared derived objects, and nonshareable derived objects)	Pathname to the data container in the view's private storage area
Version whose element uses a single data container file	Pathname to the data container in the VOB cleartext storage pool NOTE: If you have accessed the version recently, this is the actual pathname of the data container. If you have not accessed the version recently, this is the pathname to which ClearCase would extract the version.
Version whose element uses a separate data container file for each version	Pathname to the data container in the VOB source storage pool

mvfsstorage is intended for use in finding discrepancies in OS-level access rights between the view and the underlying MVFS storage. Usually, such discrepancies occur as a result of **set-UID root**, or creating files as *root* user. If you encounter a permissions error that seems unfounded, run this utility as a diagnostic.

OPTIONS AND ARGUMENTS

pname ...

One or more names of files whose pathnames are under a VOB-tag (an *MVFS object*). For directories and non-MVFS objects, **mvfsstorage** echos the pathnames you give it.

EXAMPLES

- For a view-private file, compare view-level ownership and permissions against those on the file's underlying storage location.

```
% ls -l unixV7 '/usr/atria/etc/mvfsstorage unixV7'
-rwxrwxrwx  1 nobody 65534  2210032  May 12 09:33 /net/myhost/home/myview/
    .s/0008.VOB/016D.2E2F.unixV7*
-rwxrwxrwx  1 root   sys    2210032  May 12 09:33 unixV7*
```

SEE ALSO

mvfscache, **mvfslog**, **mvfsstat**, **mvfstime**, **mvfsversion**
Administering ClearCase

mvfstime

Summarizes MVFS activity while a command is executing

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

ccase-home-dir/**etc/mvfstime** [**–icrvVhalzAd**] [**–o** *outfile*] [*time*] [*count*] *command* [*args*]

DESCRIPTION

The **mvfstime** command executes a command and sends to **stderr** a report consisting of:

- Standard UNIX Timing statistics
- MVFS usage statistics, similar to those generated by **mvfsstat**.

Use this command to perform timing experiments for applications running in a ClearCase environment.

IMPORTANT: The statistics gathered while *command* is running under **mvfstime** are systemwide statistics for that time period and are not limited to that command's activities. To get an accurate reading of the MVFS activity of *command*, make sure that no other activity is taking place on the machine when you invoke **mvfstime**.

See the **mvfsstat** reference page for an explanation of MVFS statistics. See the **csh(1)** reference page for information on UNIX statistics.

OPTIONS AND ARGUMENTS

See the **mvfsstat** reference page for a description of the command-line options.

EXAMPLES

- Generate timing statistics for an invocation of the **clearmake** program.

```
% /usr/atria/etc/mvfstime -iclr clearmake
----------------- Started at Mon Jul 19 10:58:48 1999 -------------------
        cp test2.txt file2sub.txt

        cp file2sub.txt file2.txt

        cat file1.txt > foo
        cat file2.txt >> foo
```

```
----------------- Ended at Mon Jul 19 10:59:01 1999 -------------------
time:    0.7u  0.9s  0:13 13% 0+0io 0pf+0w
Directory Name Cache:     93 calls
                82    ( 88.2%) hit:
                                  24 current directory
                                   0/1      directories    (  0.0%)
                                  42/48     regular files  ( 87.5%)
                                  16/26     name not found ( 61.5%)
                11    ( 11.8%) miss
                                   0 event misses
                17    ( 18.3%) add:
                                   1 directories
                                   6 regular files
                                  10 name not found
Attribute cache:   287 calls
                274   ( 95.5%) hit: 2 lvut-generated
                13    (  4.5%) miss:
                                  10 close-to-open
                                   0 build generation mismatch
                                   0 timed out
                                   0 new
                                   3 vob/view event;
                                   0 lvut also missed
                56     updates
                                  14 unexpected modifications
                                   7 expected modifications
                                   0 VOB/view cache modifications

Cleartext I/O:
        Cleartext layer:
        -------clriotype------calls---c/s------rt--rt/call-rt%
        get                         3  0.23   0.001  0.000   0%
        create                      5  0.38   0.142  0.028   1%
        read                       19  1.46   0.017  0.001   0%
        write                      19  1.46   0.006  0.000   0%
        open                       12  0.92   0.000  0.000   0%
           clrio total:            58  4.46   0.167          1%

        MVFS layer:
        -------clriotype------calls---c/s------rt--rt/call-rt%--mvfs%
        cto_getattr                10  0.77   0.016  0.002   0%
        read                       19  1.46   0.002  0.000   0%    11%
        write                      19  1.46   0.002  0.000   0%    25%
        get/open/cto               16  1.23   0.216  0.013   2%    92%

Remote calls to view server:
```

```
          ----------rpc-------calls---c/s------rt--rt/call
          setattr                18  1.38   0.176  0.010   1%
          create                  5  0.38   0.102  0.020   1%
          rename                  2  0.15   0.132  0.066   1%
          readdir                 1  0.08   0.006  0.006   0%
          cltxt                   3  0.23   0.012  0.004   0%
          chg oid                 7  0.54   0.201  0.029   2%
          revalidate              1  0.08   0.007  0.007   0%
          ch mtype                3  0.23   0.028  0.009   0%
          lookup                 11  0.85   0.036  0.003   0%
          getattr                13  1.00   0.021  0.002   0%
          replica root            1  0.08   0.002  0.002   0%
             rpc total:          65  5.00   0.722          6%  0
retransmissions

RPC handles:
          gets             65
          creates           0 (  0% of gets)
          destroys          0
Largest excessive RPC delay: 0 seconds
```

SEE ALSO

csh(1), **mvfscache**, **mvfslog**, **mvfsstat**, **mvfsstorage**, **mvfsversion**

mvfsversion

Displays MVFS version string

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

mvfsversion [**–r**] [**–s**]

DESCRIPTION

The **mvfsversion** command displays the version string of your host's MVFS, in RCS or SCCS format. This string also appears at operating system startup.

OPTIONS AND ARGUMENTS

Default: The MVFS version string is displayed in SCCS format.

–s
Same as default.

–r
Displays the version string in RCS format.

EXAMPLES

- Display the MVFS version string in RCS format.

  mvfsversion –r
  ```
  $Header: MVFS version 4.0 (Wed Jan 21 02:15:44 EST 1999) $
  ```

SEE ALSO

mvfscache, **mvfslog**, **mvfsstat**, **mvfsstorage**, **mvfstime**

mvws

Move or rename current workspace

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

mvws *ws-dir-name*

DESCRIPTION

The **mvws** command changes the name of the current *workspace storage directory* or moves it to a new parent directory in the local file system.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE WORKSPACE'S NEW LOCATION. *Default*: None. You must specify a new location for the workspace storage directory.

ws-dir-name

If *ws-dir-name* already exists, it specifies the parent directory into which the workspace storage directory will be moved. If *ws-dir-name* does not already exist, it specifies the new name for the workspace storage directory.

EXAMPLES

- Show a listing of the current workspace, and then move its storage directory to the **bin** directory. At an Attache prompt:

  lsws

  ```
  Workspace name      Local storage directory    Server host
   jed_ws             C:\users\jo\jed_ws         agora
  ```

  mvws c:\users\jo\bin

  lsws

  ```
  Workspace name      Local storage directory    Server host
    jed_ws           C:\users\jo\bin\jed_ws     agora
  ```

- Show a listing of the current workspace, and then rename its storage directory. At an Attache prompt:

  lsws

  ```
  Workspace name      Local storage directory    Server host
    rdc_ws            C:\users\rdc\rdc_ws        neon
  ```

  mvws c:\users\rdc\darren_ws

  lsws

  ```
  Workspace name      Local storage directory    Server host
    darren_ws         C:\users\rdc\darren_ws     neon
  ```

SEE ALSO

attache_command_line_interface, **mkws**, **lsws**, **rmws**

pathnames_ccase

Pathname resolution, dynamic view context, and extended namespace

APPLICABILITY

Product	Command Type
ClearCase	general information
ClearCase LT	general information

SYNOPSIS

- VOB-extended pathname:

  Element: *element-pname@@*
  Branch: *element-pname@@branch-pname*
  Version: *element-pname@@version-selector*
  VOB symbolic link: *link-pname*
  Derived object: *derived-object-pname@@derived-object-ID*

- Absolute VOB pathname (dynamic views only):

  /*vob-tag***/***pname-in-vob*

- View-extended pathname (dynamic views only):

  /view /*view-tag* **/***full-pathname*

DESCRIPTION

This reference page describes ClearCase and ClearCase LT extensions to the standard file/directory namespace provided by the operating system. These extensions can be used as follows:

- From a *dynamic view*, you can use the pathname forms described here as arguments to any **cleartool** command that takes a pathname.
- From a *snapshot view*, you can use the VOB-extended pathname forms as arguments to those **cleartool** commands that return information about elements and versions (for example, **describe**, **ls**, **lshistory**, and **diff**). Such operations do not require the *MVFS*. However, you cannot use VOB-extended pathnames forms to check out an element version that is not loaded into your view.

DYNAMIC VIEW CONTEXTS

A pathname can access ClearCase or ClearCase LT data only if it has a *view context*:

- **SET VIEW CONTEXT** — A process, typically a shell, created with the **setview** command is said to have a *set view* context. That process, along with all of its children, is "set to the view."
- **WORKING DIRECTORY VIEW CONTEXT** — You can change the current working directory of a process to a view-extended pathname:

  % cd /view/david/vobs/proj

  Such a process is said to have a *working directory view* context.(The process may or may not also have a *set view* context.)
- **VIEW-EXTENDED PATHNAME** — A pathname can specify its own view context, regardless of the current set view or working directory view contexts, if any.

KINDS OF PATHNAMES

The following sections describe the kinds of pathnames you can use with ClearCase and ClearCase LT.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Standard Pathnames

A standard pathname is either full or relative:

- A *full pathname* begins with a slash (**/**):

  /usr/bin/cc

  The slash can be implied. For example:

  ~bertrand/proj/doc

  A full pathname is interpreted in the process's set view context. An error occurs if you attempt to use a full pathname to access ClearCase data in a process that is not set to a view.
- A *relative pathname* does not begin with a slash:

  foo.c

  ../motif/libX.a

A relative pathname is interpreted in the process's working directory view context, if it has one. Otherwise, it uses the process's set view context. If a process has neither kind of view context, an error occurs.

A standard pathname can reference any kind of file system object: For example, **/vobs/proj/BAR** references "file-system object named 'BAR', as seen through the current view." This can be any of the following:

- **VERSION** — If **BAR** names an element, the pathname references the version of that element selected by the current view's config spec.
- **VOB SYMBOLIC LINK** — **BAR** can name a VOB symbolic link that is visible in the current view. Depending on the command, the link may or may not be traversed.
- **DERIVED OBJECT** — **BAR** can name a derived object that was built in the current view or was winked in to the view.
- **VIEW-PRIVATE OBJECT** — **BAR** can name a view-private object (including a checked-out version) located in the current view's private storage area.
- **NON-MVFS OBJECT** — **BAR** can name an object that is not under ClearCase or ClearCase LT control, such as objects in your home directory or in **/usr/bin**.

Using standard pathnames to reference MVFS objects is termed *transparency*: a view's **view_server** process resolves the standard pathname into a reference to the appropriate MVFS object. In essence, transparency makes a VOB appear to be a standard directory tree.

Extended Pathnames

The MVFS supports two kinds of extensions to the standard pathname scheme:

- You can add two pathname components to the beginning of any full pathname, turning it into a *view-extended pathname*:

  /view/david/vobs/proj/foo.c *(view-extended full pathname)*

- In certain situations, a relative pathname can include a view specification:

  ../../david/vobs/proj/foo.c *(view-extended relative pathname)*

- You can add characters to the end of a relative or full pathname, turning it into a *VOB-extended pathname*. VOB-extended pathnames that specify versions of elements are the most commonly used; they are termed *version-extended pathnames*.

  foo.c@@/main/12 *(version-extended pathname)*

  /vobs/proj/foo.c@@/main/motif/4 *(version-extended pathname)*

  foo.c@@/RLS4.3 *(version-extended pathname)*

  foo.c@@/main *(VOB-extended pathname to a branch)*

  foo.c@@ *(VOB-extended pathname to an element)*

  hello.o@@15-Sep.08:10.439 *(VOB-extended pathname to a derived object)*

VIEW-EXTENDED PATHNAMES

A *view-extended pathname* is a standard pathname, along with a specification of a view. For example, **/view/david/vobs/proj/BAR** references file-system object named **BAR**, as seen through view **david**. A view-extended pathname can access any kind of file-system object, as described in *Standard Pathnames* on page 729.

The Viewroot Directory / View-Tags

In most view-extended pathnames, a full pathname is prepended with two components: the name of the host's *viewroot directory* and the *view-tag* of a particular view. The *viewroot* directory is a virtual data structure, whose contents exist only in MVFS buffers in main memory. Each view is made accessible to standard programs and ClearCase programs through a *view-tag* entry in the viewroot directory. No standard command or program can modify this directory. Only a few ClearCase commands use or modify it: **mkview**, **mktag**, **rmtag**, **rmview**, **startview**.

The viewroot directory is activated by a standard **mount(1M)** command, which considers the virtual data structure to be a file system of type MVFS. The ClearCase pathname of the viewroot directory is **/view**. See the **init_ccase** reference page and the **view** reference page for details.

SYMBOLIC LINKS AND THE VIEW-EXTENDED NAMESPACE

Pathnames are resolved component-by-component by the operating system kernel and the MVFS. When a UNIX symbolic link or VOB symbolic link is traversed, a full pathname needs a set view context to access ClearCase data. Thus, a symbolic link whose text is a full pathname such as

```
/vobs/aardvark -> /vobs/all_projects/aardvark ...
```

is interpreted in the current set view context. If the process has no set view context, traversing such a symbolic link will fail.

VOB-EXTENDED PATHNAMES

The transparency feature enables you to use standard pathnames to access version-controlled data; the **view_server** does the work of locating the data. But you can also bypass transparency and do the work yourself:

- You can access any version of an element by using its *version-ID*, which specifies its exact version-tree location:

  sort.c@@/main/motif/4

- If a version has been assigned a version label, you can access it using the label:

  sort.c@@/main/motif/RLS_1.3 *(branch and version label)*
  sort.c@@/RLS_1.3 *(version label only)*

Typically, you can use the label, without having to specify the branch on which the labeled version resides; see *Version Labels in Extended Namespace*.

- You can access any element object or branch object directly:

  sort.c@@ *(element object)*
  sort.c@@/main *(branch object)*
  sort.c@@/main/motif *(branch object)*

- You can access any derived object directly, regardless of the view it was created in:

  sort.o@@13-Aug.09:45.569 *(derived object created on 13-Aug)*
  sort.o@@23-Sep.19:09.743 *(derived object created on 23-Sep)*

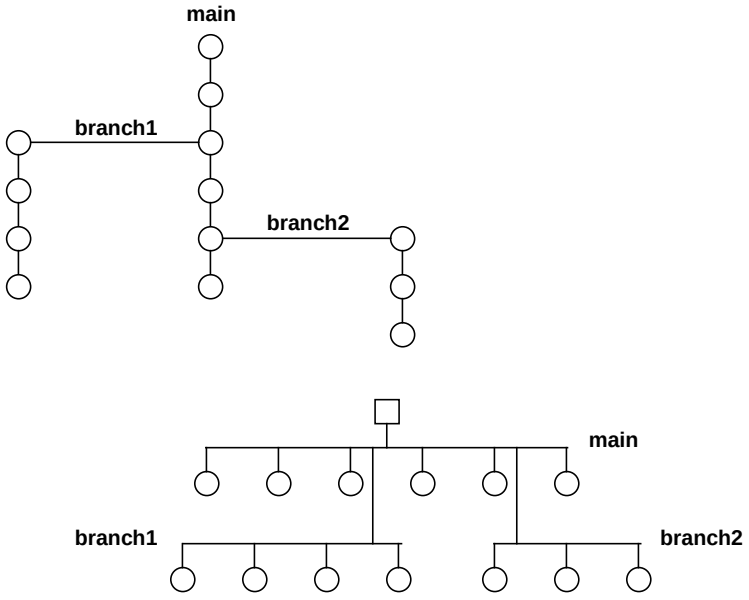
The pathnames in the above examples are termed *VOB-extended pathnames*. A VOB's file/directory namespace is extended in two ways from the standard namespace: one extension enables direct access to elements, branches, and versions; the other enables direct access to derived objects. Both extensions allow you to access objects not visible in your own view (and perhaps not currently visible in any other view, either).

Extended Namespace for Elements, Branches, and Versions

An element's version tree has the same form as a standard directory tree (see Figure 12).

Component of Version Tree	Component of Directory Tree in Extended Namespace
element	Root of tree: The element itself appears to be a directory, which contains a single subdirectory, corresponding to the **main** branch. (It can also contain some version label; see *Version Labels in Extended Namespace*.)
branch	Subdirectory: Each branch appears to be a directory, which contains files (individual versions and version labels), directories (subbranches), and links (version labels).
version	Leaf name: Each version appears to be a leaf of a directory tree. For a file element, the leaf contains text lines or binary data, and can be processed with standard commands like **cat**, **diff**, and **cmp**. For a directory element, the leaf contains a directory structure, and can be processed with standard commands like **ls** and **cd**.

Figure 12 Version Tree and Extended Namespace



Accordingly, any location within an element's version tree can be identified by a pathname in this extended namespace:

sort.c@@	*(specifies an element)*
sort.c@@/main	*(specifies a branch)*
sort.c@@/main/branch1	*(specifies a branch)*
sort.c@@/main/branch1/2	*(specifies a version)*
doctn/.@@/main/3	*(special case: extra component is required in VOB's top-level directory)*

Extended Naming Symbol

The previous pathname examples incorporate the *extended naming symbol* (@@). This symbol is required to effect a switch from the standard file/directory namespace to the extended element/branch/version namespace. There are two equivalent ways to think of @@

- When appended to the name of any element, the extended naming symbol turns off transparency (automatic version-selection). Thus, you must specify one of the element's versions explicitly.

- The extended naming symbol is part of an element's official name. For example, **foo.c** is the name of a version (the particular version that appears in the view); **foo.c@@** is the name of the element itself.

NOTE: The establishment of @@ as the extended naming symbol occurs at system startup time with a file system table entry. Thus, different symbols may be used on different hosts. See the **init_ccase** reference page for details.

Version Labels in Extended Namespace

Version labels appear in the extended namespace as hard links. If version **/main/4** of an element is labeled **RLS_1**, the extended namespace directory corresponding to the element's **main** branch lists both **4** and **RLS_1** as hard links to the version:

```
% ls -il sort.c@@/main
246 -r--r--r-- 1 drp user 217 Oct 6 21:12 4
.
.
.
246 -r--r--r-- 1 drp user 217 Oct 6 21:12 RLS_1
```

If the label type was created with the once-per-element restriction, an additional hard link to the labeled version appears in the element's top-level directory:

```
% ls -il sort.c@@
246 -r--r--r-- 1 drp user 217 Oct 6 21:12 RLS_1
```

In this case, all the following are equivalent extended pathnames to the labeled version:

sort.c@@/RLS_1	*(version label at top level of element)*
sort.c@@/main/4	*(version-ID)*
sort.c@@/main/RLS_1	*(version label at branch level)*

(The once-per-element restriction is the **mklbtype** default. A **mklbtype –pbranch** command creates a label type that can be used once on each branch of an element.)

Pathnames Involving More Than One Element

A VOB can implement a deep directory structure. Thus, a pathname can involve several elements. For example:

/vobs/proj/src/include/sort.h

If **proj** is the VOB's root directory element, then **src** and **include** also name directory elements, and **sort.h** names a file element.

After a pathname crosses over into the extended namespace with **@@,** you must specify a version for each succeeding element in the pathname. For example:

/vobs/proj/src/include@@/main/4/sort.h/main/LATEST

To automatically select versions for elements **proj** and **src**: cross over to extended namespace at directory element **include**, specifying a version of **include** and a version of **sort.h**:

/vobs/proj/src@@/RLS_1/include/RLS_1/sort.h/RLS_1

To automatically select versions for element **proj** only: cross over to extended namespace at directory element **src**, specifying the version labeled **RLS_1** of each succeeding element:

/vobs/proj@@/main/1/src/main/4 *(invalid)*
/vobs/proj/.@@/main/1/src/main/4 *(valid)*

SPECIAL CASE: When crossing over into extended namespace at the VOB root directory (that is, at the VOB-tag or VOB mount point), you must use **/.@@** instead of **@@**.

The extended naming symbol need be used only once in a pathname, to indicate the crossover into extended namespace. You can, however, append it to any element name:

/vobs/proj/src@@/RLS_1/include@@/RLS_1/sort.h@@/RLS_1

Reading and Writing in the Extended Namespace

A VOB-extended pathname references an object in a VOB database. The reference can either read or write the database—that is, either query metadata or modify metadata:

% **cleartool mklabel RLS2.1 util.c@@/RLS2.0** *(attach an additional label to a version)*

% **cleartool rmattr BugNum util.c@@/main/3** *(remove an attribute)*

For a version, an extended pathname can also read the version's data, but cannot write or delete it:

% **grep 'env' util.c@@/main/rel2_bugfix/1** *(valid)*

% **rm util.c@@/main/rel2_bugfix/1** *(invalid)*

```
ERROR: util.c@@/main/rel2_bugfix/1 not removed: Read-only file system.
```

Extended Namespace for Derived Objects

The extended namespace allows multiple derived objects to exist at the same standard pathname. Multiple versions of an element also exist at the same standard pathname, but the two extensions work differently. Derived objects created at the same location are distinguished by their unique derived object identifiers, or *DO-IDs*:

```
sort.o@@14-Sep.09:54.418
sort.o@@13-Sep.09:30.404
sort.o@@02-Sep.16:23.353
.
.
.
```

An extended name provides access only to the derived object's metadata in the VOB database—principally, its configuration record. DO-IDs can be used only with ClearCase commands; they cannot be used in non-ClearCase programs (for example, editors or compilers).

Navigating the VOB-Extended Namespace

You can use standard directory-navigation commands (**cd**, **ls**, **pwd**, and so on) in a VOB's extended namespace. For example, these are two equivalent ways to display the contents of an old version:

- Use a version-extended pathname from a standard directory:

  % **cat util.c@@/main/rel2_bugfix/1**

- Change to branch "directory" in the VOB-extended namespace, and then display the version:

  % **cd util.c@@/main/rel2_bugfix**

  % **cat 1**

In VOB-extended namespace, elements and branches are directories; you can change to such directories with **cd**; you can lists their contents—branches and versions—with **ls**.

You can access versions of file elements as ordinary files, with **cat**, **diff**, and so on—even executing versions that happen to be compiled programs or scripts.

SPECIAL VIEW-TAG REPORTED BY PWD. When you have changed to a VOB-extended namespace directory, the **pwd(1)** command reports your current working directory as under a special view-tag: For example:

```
% cd /view/akp_vu/vobs/proj/special@@

% pwd
/view/akp_vu@@/vobs/proj/main/4/special
```

The special view-tag **akp_vu@@** appears as a separate entry from **akp_vu** in your host's viewroot directory. When in the context of a special view-tag, version-selection is suppressed completely. To access a particular version of any file or directory element, you must specify the version explicitly. These special entries are periodically deleted on a least-recently-used basis.

EXITING FROM VOB-EXTENDED NAMESPACE. To exit VOB-extended namespace, change to a standard full pathname or a view-extended pathname. (The pathname can specify a VOB or non-VOB location.) For example:

```
% cd /vobs/proj/src@@/main                    (enter VOB-extended namespace)
% pwd                                         /view/david@@/vobs/proj/main/4/src/main
/view/david@@/vobs/proj/main/4/src/main
% cd /vobs/proj                               (exit VOB-extended namespace)
% pwd
/vobs/proj
```

Repeated use of **cd ..** does not work as you may expect. You do not exit extended namespace where you entered it; instead, you ascend through all the extended-namespace directories listed by **pwd**. For example:

```
% cd util.c@@/main/rel2_bugfix

% ls
0       1       2       LATEST

% pwd
/view/drp_fix@@/usr/hw/main/1/src/main/2/util.c/main/rel2_bugfix

% cd ../../..

% pwd
/view/drp_fix@@/usr/hw/main/1/src/main/2

% cd ../..

% pwd
/view/drp_fix@@/usr/hw/main/1/src

% cd ../../../

% pwd
/view/drp_fix@@/usr/hw
```

SEE ALSO

query_language, **version_selector**, **wildcards_ccase**

permissions

Permissions checking

APPLICABILITY

Product	Command Type
ClearCase	general information
ClearCase LT	general information
Attache	general information
MultiSite	general information

DESCRIPTION

In general, only commands that modify (write to) a VOB are subjected to permissions checking. The following hierarchy (in order from most privileged to least privileged) is used, in a command-specific manner, to determine whether a command can proceed or be canceled:

- **root**
- *VOB owner*
- Owner of the relevant element (for modifications to branches and versions)
- Owner of the relevant type object (for modifications to objects of that type)
- Creator of a version or derived object
- Owner of the object (pool, hyperlink, replica, activity, checkpoint, domain, role, state, user)
- User associated with an event
- Members of an object's group (same UNIX group-ID)

For example, **root** always has permission to use commands that modify a VOB. However, if you try to modify an element that you do not own, and you are not the VOB owner or **root**, the command fails.

Both file-system and non-file-system objects have an owner and a group; this information is stored in the **stat** record of the object. When an object is created, its owner and group are set to that of the user who created it. Use the **protect** command to change the owner (**–chown**) or group (**–chgrp**) of the object. The **describe** command displays the owner and group of the object.

The scheduler maintains its own access control list (ACL),which determines who is allowed access to the scheduler and to the ACL itself. See the **schedule** reference page for more information.

The reference page for a command lists the permissions required to use the command.

The sections below list all **cleartool** subcommands and Attache commands, categorized by their permissions requirements.

None

annotate
apropos
catcr
catcs
cd
checkvob (except with **–fix** or **–hlink**)
describe
diff
diffcr
dospace [1]
edcs
endview (except with **-server**)
file
find
findmerge [2]
get
getcache
getlog
help
hostinfo
import [3]
ln [4]
ls
lsactivity
lscheckout
lsclients
lsdo
lshistory
lslocal
lslock
lsmaster
lspool
lsprivate
lsregion
lsreplica
lssite
lsstgloc
lstype
lsview
lsvob
lsvtree
lsws
make
man
mkattype [5]
mkbrtype [5]
mkdir [4]
mkelem [4]
mkeltype [5]
mkhltype [5]
mklbtype [5]
mkregion
mkstgloc
mktag [6]
mkview [7]
mkvob [7]
mkws
mount [10]
mv [4]
mvws
put
pwd
pwv
quit
recoverview
reformatview
register
reqmaster (requesting mastership only) [9]
rmname [4 8]
rmregion
rmstgloc
rmtag
rmws
setcs
setsite
setview
setws
shell
space [1]
startview
umount (public VOB)
unregister
update
winkin
wshell

[1] Except with **–update** or **–generate**
[2] No permissions required for "search" functionality
[3] For created elements only
[4] One or more directory elements must be checked out
[5] Except with **–replace**
[6] Except for private VOB-tag
[7] tandard UNIX permissions for creating a subdirectory required
[8] Except with **–nco**
[9] Must be on ACL at master replica
[10] Only for public VOB

one of: element group member, element owner, VOB owner, root;
(for commands that operate on objects) object group member, object owner , VOB owner, root

checkout	**mkattr**	**mktrigger**	**rmlabel**
checkvob –hlink	**mkbranch**	**reserve**	**rmmerge**
import [1]	**mkhlink**	**rmattr**	**rmtrigger**
merge [2]	**mklabel**	**rmhlink**	**unreserve**

[1] For checked-out directories only

[2] Applies to creation of merge arrows only, not to data

one of: version creator, element owner, VOB owner, root

checkin	**uncheckout**
rmver	

one of: element owner, VOB owner, root

chtype (element)	**rmelem**
lock (element)	**unlock** (element)

one of: user associated with event, object owner, VOB owner, root

chevent

one of: branch creator, element owner, VOB owner, root

chtype (branch)	**rmbranch**
lock (branch)	**unlock** (branch)

one of: type owner, VOB owner, root

lock (type object)
mkattype –replace
mkbrtype –replace
mkeltype –replace
mkhltype –replace
mklbtype –replace
mktrtype –replace
rename (type object)
rmtype
unlock (type object)

one of: pool owner, VOB owner, root

rename (pool)
rmpool

one of: DO group member, DO owner, VOB owner, root

rmdo

NOTE: Only the VOB owner and **root** can delete a shared derived object.

one of: view owner, root

endview -server
rmview
setcache –view
space –view –generate

one of: owner, VOB owner, root

chmaster
chuser
protect

one of: VOB owner, root

checkvob –fix
chpool
dospace –generate
ln –nco
lock (pool or VOB)
mkpool
mktrtype [1]
reformatvob
relocate
reqmaster (to set access controls)
rmname –nco
rmvob
space –vob –generate
umount (private VOB)
unlock (pool or VOB)

[1] except with **–replace**

VOB owner

mktag (private VOB-tag)
mount (private VOB)

view owner

chview (can also be **root** on view server host)

root

protectvob **setcache –mvfs**
setcache –host

same permissions as for creating the type object with a mktype command**

cptype

permissions controlled by the scheduler ACL

dospace –update **space –update**
schedule

SEE ALSO

Reference pages for individual commands

profile_ccase

cleartool user profile: **.clearcase_profile**

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure

SYNOPSIS

command_name flag
.
.
.

DESCRIPTION

The **cleartool** user profile (**.clearcase_profile**) is an ordered set of rules that determine certain command option defaults for one or more **cleartool** commands. An option you supply in a command line overrides the command option default specified in **.clearcase_profile**.

For example, many **cleartool** commands accept user comments with the **–c**, **–cfile**, **–cq**, **–cqe**, or **–nc** option. If you specify none of these options, **cleartool** invokes one of them by default. The option invoked varies from command to command, but is always one of **–cq**, **–cqe**, or **–nc**. If **cleartool** finds a file named **.clearcase_profile** in your home directory, it checks to see whether it contains a comment rule that applies to the current command. If so, it invokes the comment option indicated by that rule. No error occurs if this file does not exist; **cleartool** invokes the command's standard comment default.

An alternate name for the user profile can be specified with the environment variable **CLEARCASE_PROFILE**. Its value should be a full pathname.

HOW cleartool SELECTS A RULE

For a given command, **cleartool** consults the user profile to determine which rule, if any, applies to a command. The method is similar to the one used by the **view_server** process to evaluate a config spec:

- **cleartool** examines the first rule in the user profile and decides whether it applies to the specified command.
- If the rule does not apply, **cleartool** goes on to the next rule in the file; it repeats this step for each succeeding rule until the last.

- If no rule applies, **cleartool** invokes the standard default for the command option.

cleartool uses the first rule that applies. Therefore, the order of rules in the user profile is significant. For example, to ensure that you are always prompted for a comment when you create a directory element, you must place a rule for the **mkdir** command before any more general rule that may also apply to **mkdir**, such as *** –nc**.

RULE SYNTAX

Rules must be placed on separate lines. Extra white space (space, tab) is ignored.

Comments begin with a number sign (**#**). For example:

```
#element rules
mkelem -cqe     #prompt for comment for each new element being created
```

Each rule consists of two tokens, separated by white space:

command_name *flag*

COMMENT RULES

When specifying a comment rule:

- *command_name* must be one of these or an asterisk (*****), which matches all of them:

checkin	**mkdir**	**mklbtype**
checkout	**mkelem**	**mkpool**
mkattype	**mkeltype**	**mktrtype**
mkbrtype	**mkhltype**	**mkvob**

- *flag* must be one of these: **–nc**, **–cqe**, **–cq**. The **–c** and **–cfile** options are not valid here.

If you do not provide a comment rule for one of these commands, **cleartool** uses **–cqe** as its default comment option. **cleartool** uses **–nc** as the default for all other commands that accept comments.

RULES FOR CHECKED-OUT VERSION STATES

When specifying a rule for the state of a checked-out version:

- *command_name* must be **checkout**.
- *flag* must be **-reserved** or **-unreserved**.

If one rule only is specified, all checkouts are reserved or unreserved by default. If the rules are specified as

checkout -reserved
checkout -unreserved

then a reserved checkout is attempted. If there is a conflict, an unreserved checkout is performed.

RULE FOR INTERACTIVE RESOLUTION OF CHECKOUT PROBLEMS

When specifying the rule for the interactive resolution of checkout problems:

- *command_name* must be **checkout**.
- *flag* must be **-query**.

When this rule is specified, you are queried about how to proceed when **checkout** encounters certain kinds of checkout problems.

EXAMPLES

- Never prompt for a comment.

  *** –nc**

- During a checkin operation, prompt for a comment for each element. During a make directory operation, prompt for a single comment to be applied to all the new directories. In all other cases, do not prompt at all.

  checkin –cqe
  mkdir –cq
  *** –nc**

- Make all checkouts unreserved.

  checkout -unreserved

- Ask how to proceed in the event of a checkout problem.

  checkout -query

SEE ALSO

checkout, **cleartool**, **comments**, **config_spec**

promote_server

Changes storage location of derived object data container

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

Invoked by **clearmake** or **winkin**, if necessary, when it winks in a derived object

DESCRIPTION

NOTE: Never run **promote_server** manually. It must be invoked only by **clearmake**. See the **view_scrubber** reference page for information on transferring a derived object's data container to VOB storage.

The **promote_server** program migrates a *derived object*'s data container file from private storage to shared storage. When **clearmake** winks in a derived object (DO) that was previously unshared, it invokes **promote_server** to copy the data container file from view-private storage to a *VOB storage pool*.

NOTE: **clearmake** also migrates a DO's configuration record from private storage to shared storage at the same time. This work is performed by **clearmake** itself, not by **promote_server**.

The destination storage pool is determined by the DO's pathname. By definition, this pathname is under a VOB-tag; that is, the DO is in some VOB directory. The DO storage pool to which the directory element is assigned is the destination of the promotion. (Some build scripts create multiple hard links, in different directories, to a derived object. In this case, the data container is promoted to the storage pool of only one of the directories.)

clearmake invokes **promote_server** by making a request to the ClearCase master server, **albd_server**. **promote_server** runs as the owner of the view in which the data container to be copied resides. This ensures that the data container is readable.

After promoting a DO, the **promote_server** remains active for several minutes to ensure that subsequent promotions from the same view are processed with the least overhead. During this time, the **promote_server** remains associated with the view from which the DO was promoted; if two users try to promote DOs from the same view, at the same time, they share (serially) the same **promote_server**.

SEE ALSO

albd_server, **clearmake**, **view_scrubber**, **view_server**

protect

Changes permissions or ownership of a VOB object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

protect [**–cho·wn** *login-name*] [**–chg·rp** *group-name*] [**–chm·od** *permissions*]
[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
{ [**–fil·e** | **–d·irectory**] [**–r·ecurse**] [**–pna·me**] *pname* ...
| *object-selector* ...
}

DESCRIPTION

The **protect** command sets the owner, group, or permissions for one or more elements, shared derived objects, or named VOB objects. This information is maintained in the VOB database.

NOTE: This command does not apply to files loaded in a *snapshot view*.

The main use of **protect** is to control access by standard programs to an element or object's data. For example, you may make some elements readable by anyone, and make others readable by only their group members.

Modifying the permissions of an element changes the permissions of all of its source containers and (if applicable) cleartext containers. That is, the change affects all versions, not just the version selected by the current view. There is no way to change the permissions of an individual version.

Some forms of **protect** affect ClearCase and ClearCase LT access. For example, a checkout or checkin is permitted only if the user is the element's owner, or is a member of the element's group.

View-Private Objects

This command does not affect view-private objects. For this reason, entering a **protect** command sometimes seems to have no effect:

- Changing an element's permissions has no effect on its checked-out versions. After you check in the element, your view selects the checked-in version, thus making the updated permissions appear.
- Changing a DO's permission has no effect on the way the DO appears in the view where it was originally created, or in the dynamic views where it has been winked in. To have your dynamic view use a shared DO with updated permissions:
  a. Use **protect** to change the permissions on the DO in the VOB database.
  b. Use **rm** to remove the DO from your view.
  c. Use **clearmake** or the **winkin** command to wink in the DO, with its new permissions.

You can change the permissions on any view-private object (including a checked-out version), with the standard UNIXcommands.

NOTE: We support the BSD semantics (POSIX CHOWN RESTRICTED) for **chown**: only **root** can change the owner-IDs. (In a view, this means the **root** identity on the machine on which the view storage directory resides.)

A winked-in DO is not really a view-private object, but it behaves like one (so that users in different views can build software independently). Moreover, changing the permissions of a winked-in DO actually converts it to a view-private file in your view. See *Building Software with ClearCase*.

Owner Setting

The initial owner of an element is the user who creates it with **mkelem** or **mkdir**. The initial owner of a named VOB object is the user who creates it. The initial owner of a derived object is the user who builds it with **clearmake**. When the derived object is *winked in* and becomes shared, its data container is promoted to a VOB storage pool. This process preserves the derived object's ownership, no matter who performs the build that causes the winkin.

See the **permissions** reference page for a list of operations that can be performed by an element's owner.

Group Setting

The initial group of an element or named VOB object is the principalgroup of its creator (the group listed in the creator's password entry).The new group specified in a **protect –chgrp** command must be one of the groups on the VOB's group list.

See the **permissions** reference page for a list of operations that can be performed by members of an element's or derived object's group.

NOTE: When you execute **protect –chgrp**, the set-UID and set-GID bits of the file mode (04000 and 02000, respectively) are always cleared. This differs from standard UNIX usage, where clearing occurs only when a non-**root** identity runs the **chgrp** command.

Read and Execute Permissions

The read and execute permissions of an element or shared derived object control access to its data in the standard manner. The permissions are also applied to all its associated data containers.

NOTE: **protect** sometimes adds group-read permission to your specification. This ensures that the owner of an element always retains read permission to its data containers.

Write Permission

The meaning of the write permission varies with the kind of object:

- For a file element, write permission settings are ignored. To obtain write permission to a file element, you must check it out (see the **checkout** reference page).
- For a directory element, write permission allows view-private files to be created within it. ClearCase or ClearCase LT permissions control changes to the directory element itself. (See the **permissions** reference page.)
- For a shared derived object, write permission allows it to be overwritten with a new derived object during a target rebuild. (The shared derived object is not actually affected; rather, the view sees the new, unshared derived object in its place.)

Protection of Global Types and Local Copies

Changing the protection of a global type or a local copy of a global type changes the protection of the global type and all its local copies. You must have permission to change the protection of the global type.

If the protection cannot be changed on one or more of the local copies, the operation fails and the global type's protection is not changed. You must fix the problem and run the **protect** command again.

For more information, see *Administering ClearCase*.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: owner, VOB owner, or **root**. See the **permissions** reference page.

NOTE: With **protect –chgrp,** you must be a member of the new group, and it must also be in the VOB's group list.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, pool (non-directory elements only). For named objects, an error occurs if the VOB, object, or object's type is locked.

Mastership Checking: (replicated VOBs only) If the current replica is ownership-preserving, an error occurs if the current replica does not master the object being processed. If the current replica is non-ownership-preserving, no mastership restrictions apply.

OPTIONS AND ARGUMENTS

SPECIFYING PERMISSION CHANGES. *Default:* None.

–cho·wn *login-name*
New owner for the elements or VOB objects, in **chown(1)** format. The owner may be either a decimal user-ID or a login name found in the **passwd**(4) file.

–chg·rp *group*
New group for the elements or VOB objects, in **chgroup(1)** format. The group may be either a decimal group-ID or a group name found in the **group(4)** file.

–chm·od *permissions*
New permissions—owner, group, other (world)—for the elements or VOB objects. Both symbolic and absolute codes are valid, such as **go–x** (symbolic) or **666** (absolute); see **chmod(1)** for details.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE OBJECTS. *Default:* None.

–fil·e
Restricts the command to changing file elements only. This option is especially useful in combination with the **–recurse** option.

–d·irectory
Restricts the command to changing directory elements only. This option is especially useful in combination with the **–recurse** option.

[–pna·me] *pname ...*
One or more pathnames, each of which specifies an element or shared derived object. If *pname* has the form of an object selector, you must use the **–pname** option to indicate that *pname* is a pathname. An extended pathname to a version or branch is valid, but keep in mind that **protect** affects the entire element. Shared derived objects can be referenced by DO-ID.

If you specify multiple *pname* arguments, but you do not have permission to change the permissions on a particular object, **protect** quits as soon as it encounters this error.

object-selector ...
One or more named VOB objects. Specify *object-selector* in one of the following forms:

attribute-type-selector **attype:***type-name*[**@***vob-selector*]

branch-type-selector	**brtype:***type-name*[**@***vob-selector*]
element-type-selector	**eltype:***type-name*[**@***vob-selector*]
hyperlink-type-selector	**hltype:***type-name*[**@***vob-selector*]
label-type-selector	**lbtype:***type-name*[**@***vob-selector*]
trigger-type-selector	**trtype:***type-name*[**@***vob-selector*]
pool-selector	**pool:***pool-name*[**@***vob-selector*]
hlink-selector	**hlink:***hlink-id*[**@***vob-selector*]
oid-obj-selector	**oid:***object-oid*[**@***vob-selector*]

The following object selector is valid only if you use MultiSite:

replica-selector	**replica:***replica-name*[**@***vob-selector*]

PROCESSING OF DIRECTORY ELEMENTS. *Default:* Any *pname* argument that specifies a directory causes the directory element itself to be changed.

–r·ecurse
Changes the entire tree of elements including and below any *pname* argument specifying a directory element. *VOB symbolic links* are not traversed during the recursive descent. (Use **–file** or **–directory** to restrict the changes to one kind of element.)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Add read permission to the file element **hello.c**, for all users.

  cmd-context **protect –chmod +r hello.c**
  ```
  Changed protection on "hello.c".
  ```

- Change the group-ID for all elements in the **src** directory to **user**.

  cmd-context **protect –recurse –chgrp user src**
  ```
  Changed protection on "src".
  Changed protection on "src/cm_fill.c".
  Changed protection on "src/convolution.c".
  Changed protection on "src/hello.c".
  Changed protection on "src/msg.c".
  Changed protection on "src/util.c".
  ```

- Change the owner of the branch type **qa_test** to **tester**.

cmd-context **protect –chown tester brtype:qa_test**
```
Changed protection on "qa_test".
```

- Allow users in the same group to read/write/execute the shared derived object **hello**, but disable all access by others. Use an absolute permission specification.

  cmd-context **protect –chmod 770 hello**
  ```
  Changed protection on "hello".
  ```

SEE ALSO

protectvob, **chmod(1)**, **chown(1)**, **chgrp(1), passwd**(4), **group**(4)

protectvob

Changes owner or groups of a VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

protectvob [**–f·orce**] [**–cho·wn** *user*] [**–chg·rp** *group*]
[**–add·_group** *group*[,...] [**–del·ete_group** *group*[,...]]
vob-storage-pname ...

DESCRIPTION

Before executing this command, log in to the host where the VOB storage directory resides, as **root** . Execute this command only when there are no active users of the VOB; it stops and restarts the associated **vob_server** process, which prevents access to storage pools.

protectvob manages the ownership and group membership of the files and directories in a VOB, by changing the OS-level permissions on files and directories within the VOB storage area. If the VOB has remote storage pools, you may need to execute this command on the remote host also to complete the permissions update. See the section *VOBs with Remote Storage Pools*.

VOB Owner and VOB Group List

A new VOB, created with **mkvob**, takes the identity of its creator:

- The creator becomes the VOB owner.
- The creator's principal group becomes the VOB's principal group.
- The creator's group list becomes the VOB's supplementary group list.

The VOB owner can perform almost any operation involving that VOB. The VOB owner owns all the VOB's data containers and storage pools. All data container manipulations are performed by a **vob_server** process, which runs with the identity of the VOB owner (see **setuid(2)**).

The VOB's supplementary group list simulates a UNIX feature that enables a user to belong to several groups.

Groups and Access Control

The VOB's set of groups controls write access on the VOB; a user's principal group must be one of the VOB's groups—principal or supplementary—for the user to create an element or derived object.

Access Control at the Individual Object Level

A VOB's owner and group list are VOB-wide settings. Similar settings are maintained at the individual object level:

- Each element in a VOB has UNIX-level access attributes:
  - User (that is, the element's owner)
  - Group (only one, not several)
  - Read-write-execute permissions (access mode)

  These attributes control access by standard UNIX programs to the element's data. For example, some elements may be made readable by anyone, while others are made readable only by group members. An element's UNIX-level attributes apply to all of its versions.
- Similarly, UNIX-level access attributes are maintained for each shared derived object in the VOB (whose data container is in a VOB storage pool).

The **protect** command controls the UNIX-level access attributes of elements and shared derived objects. An element's access attributes apply to all its source containers and (if applicable) cleartext containers.

The .identity Directory

The **cleartool describe –vob** command lists a VOB's owner and its group list. This information is recorded in subdirectory **.identity** of the VOB storage directory. See the **vob** reference page for a description of the contents of this subdirectory.

CAUTION: Do not manipulate the **.identity** directory by any means other than this command. Inconsistent settings cause errors.

Pool Protections

Each storage pool directory (**sdft**, **ddft**, **cdft**, and each user-defined pool) typically has a large number of subdirectories. **protectvob** can verify and/or change the protections of each such subdirectory, but this can be time consuming. To save time, you can have **protectvob** check only the top-level directory of each pool; if no change is required to this directory, **protectvob** does not process the pool's entire directory tree. To disable this feature, answer **yes** at the prompt:

```
Do you wish to protect the pools that appear not to need protection?
```

NOTE: If some subdirectories are owned by a different group, it has no effect on ClearCase usage. However, the subdirectories are not protected correctly; in some cases, access could be granted to users who should not have it.

VOBs with Remote Storage Pools

If any of a VOB's storage pools physically reside on a remote host (accessed through symbolic links), **protectvob** prompts you to run **protectvob** on the remote host:

```
.
.
.
cleartool: Warning: pool: "/vobstore/vega.vbs/s/s_aux01" is remote.
cleartool: Warning: Login to the remote machine "ccsvr01".
cleartool: Warning: Then run this command again, or run the chown_pool script.
```

This is necessary because when running as the *root* user, **protectvob** usually does not have the rights to change permissions in the remote storage directory. In such cases, update each remote host as follows:

1. Log in to the host as **root**.
2. Enter the **protectvob** command without specifying any options. (You can specify **–force**, if you want.) You may need to adjust the *vob-storage-pname* you specify, because it is now a remote location.

If ClearCase or ClearCase LT is not installed on the remote host, you may need to run the **chown_pool** script on the remote host, to update one or more individual storage pools there. This script is located in the ClearCase **etc** directory. See *EXAMPLES* on page 756.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be **root**. See the **permissions** reference page.

Locks: An error occurs if the VOB is locked.

Mastership Checking: (replicated VOBs only) If the current replica is ownership-preserving, an error occurs if the current replica does not master the VOB object being processed. If the current replica is non-ownership-preserving, no mastership restrictions apply.

OPTIONS AND ARGUMENTS

CONFIRMATION STEP. *Default:* **protectvob** asks for confirmation before changing the permissions in one or more storage pools.

–for·ce

Suppresses the confirmation step.

CHANGING VOB OWNERSHIP. *Default:* None. You can use **–chown** by itself, or in combination with **–chgrp**.

–cho·wn *user*
Specifies a new VOB owner. *user* can be either a login name or a numeric user-ID. That user becomes the owner of all the VOB's storage pools and all of the data containers in them.

protectvob rebuilds the **.identity** subdirectory of the VOB storage directory, reflecting the new VOB owner's user-ID, group-ID, and additional groups (if any).

–ch·grp *group*
Specifies a new principal group for the VOB. *group* can be either a group name or a numeric group-ID.

MAINTAINING THE SECONDARY GROUP LIST. *Default:* None. You can use **–add_group** and **–delete_group** singly, or together.

–add·_group *group*[,...]
Adds one or more groups to the VOB's secondary *group list*. *group* can be either a group name or a numeric group-ID.

–del·ete_group *group*[,...]
Deletes one or more groups from the VOB's secondary *group list*. *group* can be either a group name or a numeric group-ID.

SPECIFYING THE VOB. *Default:* None.

vob-storage-pname
Local pathname of a VOB storage directory.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Make user **jackson** the owner of the VOB whose storage area is **/usr/lib/vob.vb**.

cmd-context **protectvob -chown jackson /usr/lib/vob.vb**

```
This command affects the protection on your versioned object base.
While this command is running, access to the VOB will be limited.
If you have remote pools, you will have to run this command remotely.
Pool "sdft" needs to be protected correctly.
Pool "ddft" needs to be protected correctly.
Pool "cdft" needs to be protected correctly.
Protect versioned object base "/usr/lib/vob.vb"? [no] yes
Do you wish to protect the pools that appear not to need protection? [no]
no
Protecting "/usr/lib/vob.vb/s/sdft"...
Protecting "/usr/lib/vob.vb/s/sdft/0"...
Protecting "/usr/lib/vob.vb/s/sdft/1"...
...

Protecting "/usr/lib/vob.vb/d/ddft"...
Protecting "/usr/lib/vob.vb/d/ddft/0"...
...
Protecting "/usr/lib/vob.vb/c/cdft"...
Protecting "/usr/lib/vob.vb/c/cdft/2d"...
Protecting "/usr/lib/vob.vb/c/cdft/35"...
...
VOB ownership:
 owner jackson
 group user
Additional groups:
 group doc

Change the owner and group of a remote VOB storage pool.

% rlogin ccsvr01

Password:
<enter password>

% /usr/atria/etc/chown_pool jackson.user /vobaux/vega_src/s001
```

- Add one group to a VOB's group list, and remove another group.

```
cmd-context  protectvob -add_group devel -delete_group doc /usr/lib/vob.vb
This command affects the protection on your versioned object base.
While this command is running, access to the VOB will be limited.
If you have remote pools, you will have to run this command remotely.
Pool "sdft" appears to be protected correctly.
Pool "ddft" appears to be protected correctly.
Pool "cdft" appears to be protected correctly.
Protect versioned object base "/usr/lib/vob.vb"? [no] yes
Do you wish to protect the pools that appear not to need protection? [no]
no
VOB ownership:
 owner jackson
 group user
Additional groups:
 group devel
```

SEE ALSO

chpool, **mkpool**, **mkvob**, **protect**, **albd_server**, **vob**, **vob_server**

put

Uploads writable files from the workspace to the view

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

put [**–r·ecurse**] [**–compress**] [**–pti·me**] [**–to** *to-name*] [**–log** *pname*] *pname...*

DESCRIPTION

The **put** command uploads the specified writable files from the workspace to the associated *view.*

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE FILES TO BE UPLOADED. *Default:* None.

pname...

Specifies the files and/or directories to be uploaded. Wildcard patterns apply to the workspace contents; / (slash) denotes the root of the workspace.For example, **/*.c** refers to all of the **.c** files in the *workspace root*. In addition, arguments of the form **@***pname* can be used to add the contents of the local file *pname* as pathname arguments. The pathname arguments can contain wildcards, and must be listed in the file one per line, or also be of the form **@***pname*. Specifying a relative pathname for **@***pname* begins from Attache's startup directory, not the working directory, so a full local pathname is recommended.

SPECIFYING HOW THE FILES ARE TO BE UPLOADED. *Default:* When a directory is specified, its file contents are uploaded. Only writable files are uploaded, and only if the file does not exist in the view or it is different from the source file.

–to *to-name*

Specifies a destination file name or directory. If the specified destination is a directory, it becomes a prefix for each uploaded filename. If the specified destination is a file, or does not exist, then only one source argument can be specified, and it must be a file.

–pti·me

Applies the last-modified time stamp of the source file to the destination file. –**ptime** has no effect on directories.

–compress

Causes files to be compressed while being uploaded and uncompressed after the transfer to improve performance over slow communications lines. The default behavior for this option can be set with the **Preferences** command on the **Options** menu.

HANDLING OF DIRECTORY ARGUMENTS. *Default:* For each *pname* that specifies a directory element, **put** uploads the contents of that directory, but not the contents of any of its subdirectories.

–r·ecurse

Includes writable files from the entire subtree below any subdirectory *pname*. Directories are created as necessary and specified patterns are relative to the current directory.

SPECIFYING A FILE TRANSFER LOG. *Default:* None.

–log *pname*

Specifies a log file for the operation. The log file lists the workspace-relative pathname of each file transferred by the **put** command, as well as an indication of any errors that occur during the operation. Log file pathnames are absolute, not relative to the current workspace root.

Each line in a log file is a comment line, except for the names of files that were not transferred. Log files, therefore, can be used as indirect files to redo a file transfer operation.

EXAMPLES

- Upload the writable file **hello.c** to the view, naming it **hello_new.c** and preserving the time stamp. At an Attache prompt:

  put –to hello_new.c –ptime hello.c

- Upload to the view all of the writable files and subdirectories beneath the directory **src**. At an Attache prompt:

  put –r src

- Upload to the view all of the writable files listed in the file **c:\users\jed\prj_files**. At an Attache prompt:

  put @c:\users\jed\prj_files

SEE ALSO

attache_command_line_interface, **attache_graphical_interface**, **get**, **checkin**, **checkout**, **wildcards**

pwd

Prints working directory

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command
MultiSite	multitool subcommand

SYNOPSIS

pwd

DESCRIPTION

The **pwd** command lists the current working directory, as does the standard command **pwd(1)**. This command is intended for use in interactive **cleartool** and **multitool** sessions, and in shell scripts that simulate interactive sessions.

In version-extended namespace, the current working directory is listed as a pathname that is both view-extended and version-extended. (See the **pathnames_ccase** reference page.) It includes the version of each directory element between the current location and the *VOB root directory*. For example:

```
% cd util.c@@main
cmd-context pwd
/view/akp@@/usr/hw/main/1/src/main/1/util.c/main
```

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- List the name of the current working directory.

cmd-context **pwd**
```
/usr/hw
```

- (ClearCase and ClearCase LT only) Use a view-extended pathname to go to the **/usr/hw/src** directory in the context of the **jackson_old** view, and then list the name of the directory.

  ```
  % cleartool> cd /view/jackson_old/usr/hw/src

  % cleartool> pwd
  /view/jackson_old/usr/hw/src
  ```

- (ClearCase and ClearCase LT only) Change to a version-extended namespace directory and list its name. Then change back to the original directory and list its name.

  ```
  % cleartool> cd src@@

  % cleartool> pwd
  /view/jackson_vu@@/usr/hw/main/2/src

  % cd /usr/hw/src

  cleartool> cd

  cleartool> pwd
  /usr/hw/src
  ```

SEE ALSO

cd, **pwv**

pwv

Prints the working view

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only:

pwv [–s·hort] [–wdv·iew | –set·view | –root]

- ClearCase LT only:

pwv [–s·hort] [–wdv·iew | –root]

DESCRIPTION

This command does not require a product license.

The **pwv** command lists the view-tag of your current *view context*, or `** NONE **` if there is none.

Dynamic View

You can establish or change your *dynamic view* context by entering a **setview** command, or by *changing your working directory* to a view-extended pathname. If you do both, you have two view contexts:

- Your *working directory view* is used to process simple file names and relative pathnames.
- Your *set view* is used to process full pathnames, which begin with a slash (*/*).

If you change to a version-extended pathname, **pwv** adds the extended naming symbol to the view-tag (see the *EXAMPLES* section).

Snapshot View

You can establish or change your *snapshot view* context when you change to the snapshot view directory.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

LISTING FORMAT. *Default:* The annotation `Working directory view:` or `Set view:` precedes a view's view-tag.

–s·hort
Omits the annotation string. Specifying **–short** invokes **–wdview** also, unless you use **–setview**.

WORKING DIRECTORY VIEW VS. SET VIEW. *Default:* Lists both your working directory view and your set view, unless you specify **–short**.

–wdv·iew
Lists your working directory view only.

–set·view
Lists your set view only. There is no notion of a set snapshot view, so when you work in a snapshot view, the set view is always `** NONE **`.

MISCELLANEOUS

–root
Returns the root directory path of the current working view. This root is the portion of an element's absolute path that precedes the VOB tag. Use the result as the prefix for element paths in build scripts originally developed to work in a dynamic view that you now want to use in a snapshot view.

If you start a dynamic view (see **startview**) and then change to the view (rather than using setview), this option returns the extended view path. This option returns nothing when issued from a set dynamic view (see **setview**).

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

List the current set view and working directory view. In this case, they are the same.

cmd-context **pwv**
```
Working directory view: jackson_vu
Set view: jackson_vu
```

- List the working directory view only.

  cmd-context **pwv –wdview**
  ```
  Working directory view: jackson_vu
  ```

- List the current view after changing the working directory view, but before setting a view.

  % **cd /view/jackson_old/usr/hw/src**

  cmd-context **pwv**
  ```
  Working directory view: jackson_old
  Set view: ** NONE **
  ```

- (ClearCase only) List the current view after setting a view and changing the working directory view.

  cmd-context **setview jackson_vu**

  % **cd /view/jackson_old/usr/hw/src**

  cmd-context **pwv**
  ```
  Working directory view: jackson_old
  Set view: jackson_vu
  ```

- Set a dynamic view, change to a VOB directory, and then use **pwv –root**. Notice that no value is returned.

  cmd-context **setview bert_dynview_v5**

  % **cd /vobs/doc**
  cmd-context **pwv –root**
  %

- Change to a snapshot view directory and get the root of the working view directory path.

  % **cd /usr/ssview/bert_v5/vobs/doc**

  cmd-context **pwv –root**
  ```
  /usr/ssview/bert_v5
  ```

- List the current view after changing to a version-extended namespace directory. Use the short format to list the view name only.

  cd src@@

  cmd-context **pwv –short**
  ```
  jackson_vu@@
  ```

SEE ALSO

cd, **setview**, **startview**, **pathnames_ccase**

query_language

Selects objects by their metadata

APPLICABILITY

Product	Command Type
ClearCase	general information
ClearCase LT	general information
Attache	general information

SYNOPSIS

Query Primitives:

query-function **(***arg-list***)**
attribute-type-name comparison-operator value

Compound Queries:

query **&&** *query*
query **||** *query*
! *query*
(*query* **)**

DESCRIPTION

The query language is used to formulate queries on *VOBs*. It includes logical operators similar to those in the C programming language. A query searches one or more VOBs and returns the names of objects: versions, branches, and/or elements. A query may return a single object, many objects, or no objects at all.

A query primitive evaluates to `TRUE` or `FALSE`. A `TRUE` value selects an object, such as an element, branch, or version; a `FALSE` value excludes it.

A query must be enclosed in quotes if it includes spaces. You may also need to enclose a query in quotes to prevent shell-level interpretation of characters such as **(** (open parenthesis). Quoting parentheses in config specs is not required.

Queries in Version Selectors

You can use a query in a *version selector* in these contexts:

- Command-line options in the following **cleartool** commands:

  describe, **merge**, **mkattr**, **mkbranch**, **mklabel**, **rmattr**, **rmlabel**, **rmver**

- Configuration rules; see the **config_spec** reference page

- Version-extended pathnames in ClearCase, ClearCase LT, and Attache commands; see the **pathnames_ccase** reference page

A query in a version selector must be enclosed in braces ({}).

When a query is applied to a single branch, ClearCase and ClearCase LT select the most recent version on that branch that satisfies the query. For example:

cmd-context **describe -ver '/main/{attype(QAed)}' util.c**

Using a query without a *branch pathname* causes an element's entire version tree to be searched. If the query returns a single version, the version-selection operation succeeds; the operation fails if the query returns no version (`not found`) or returns more than one version (`ambiguous`). For example:

cmd-context **describe -ver '{attype(QAed)}' util.c**
```
cleartool: Error: Ambiguous query: "{attype(QAed)}"
```

Queries in the find and findmerge Commands

You can also use queries in the **find** and **findmerge** commands. In this context, the query can be enclosed in braces **({...})**. The query returns the names of all matching objects. For example:

cleartool find util.c -ver attype(QAed) -print
```
util.c@@/main/1
util.c@@/main/3
```

QUERY PRIMITIVES

The query language includes these primitives:

attribute-type-name comparison-operator value

comparison-operator is one of the following:

== != < <= > >=

Examples:

BugNum==4053
BugNum>=4000
Status!="tested"

This primitive is `TRUE` if the object itself has an attribute of that type and the value comparison is true. To test whether an object or its subobjects has a particular attribute (for example, an element or its branches and versions), use the **attr_sub** primitive.

NOTE: If no attribute named **BugNum** has been attached to an object, then **!BugNum==671** is `TRUE`, but **BugNum!=671** is `FALSE`. The second query would be true if an attribute of type **BugNum** existed, but had a different value.

attr_sub (*attribute-type-name*, *comparison-operator, value*)

With elements	TRUE if the element or any of its branches or versions has an attribute of type *attribute-type-name* that satisfies the specified comparison with *value*.
With branches	TRUE if the branch or any of its versions has an attribute of type *attribute-type-name* that satisfies the specified comparison with *value*.
With versions	TRUE if the version itself has an attribute of type *attribute-type-name* that satisfies the specified comparison with *value*.

attype (*attribute-type-name*)

With elements	TRUE if the element itself has an attribute of type *attribute-type-name*.
With branches	TRUE if the branch itself has an attribute of type *attribute-type-name*.
With versions	TRUE if the version itself has an attribute of type *attribute-type-name*.

attype_sub (*attribute-type-name*)

With elements	TRUE if the element or any of its branches or versions has an attribute of type *attribute-type-name*.
With branches	TRUE if the branch or any of its versions has an attribute of type *attribute-type-name*.
With versions	TRUE if the version itself has an attribute of type *attribute-type-name*.

brtype (*branch-type-name*)

With elements	TRUE if the element has a branch named *branch-type-name*.
With branches	TRUE if the branch is named *branch-type-name*.
With versions	TRUE if the version is on a branch named *branch-type-name*.

created_by (*login-name*)
In all cases, TRUE if the object was created by the user *login-name* (as shown by the **describe** command).

created_since (*date-time*)
In all cases, TRUE if the object was created since *date-time*. The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**
where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** | **yesterday** | **Sunday** | ... | **Saturday** | **Sun** | ... | **Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]

month := **January** | ... | **December** | **Jan** | ... | **Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want to resolve the time to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, Greenwich Mean Time (GMT) is used. (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

eltype (*element-type-name*)
In all cases, TRUE if the element to which the object belongs is of type *element-type-name*.

hltype (*hlink-type-name*)
hltype (*hlink-type-name* **, ->**)
hltype (*hlink-type-name* **, <-**)
In all cases, TRUE if the object is either end of a hyperlink (first form) named *hlink-type-name*, or is the from-end of a hyperlink (second form), or is the to-end of a hyperlink (third form)

lbtype (*label-type-name*)
In all cases, TRUE if the object itself is labeled *label-type-name*. (Because elements and branches cannot have labels, this primitive can be true only for versions.)

lbtype_sub (*label-type-name*)

With elements	TRUE if the element has a version that is labeled *label-type-name*.
With branches	TRUE if the branch has a version that is labeled *label-type-name*.
With versions	TRUE if the version itself is labeled *label-type-name*.

merge (*from-location* **,** *to-location*)
In all cases, TRUE if the element to which the object belongs has a merge hyperlink (default name: **Merge**) connecting the *from-location* and *to-location*. You can specify either or both locations with a branch pathname or a version selector. Specifying a branch produces TRUE if the merge hyperlink involves any version on that branch. The branch pathname must be complete (for example, **/main/rel2_bugfix**, not **rel2_bugfix**).

pool (*pool-name*)
In all cases, TRUE if the element to which the object belongs has a *source pool* or *cleartext pool* named *pool-name*.

trtype (*trigger-type-name*)
In all cases, TRUE if the element to which the object belongs has an attached or inherited trigger named *trigger-type-name*.

version (*version-selector*)

With elements	TRUE if the element has a version with the specified *version-selector*.
With branches	TRUE if the branch has a version with the specified *version-selector*.
With versions	TRUE if the version itself has the specified *version-selector*.

Note that in this context, *version-selector* cannot itself contain a query. For example, **version(REL1)** is valid, but **version(lbtype(REL1))** is not.

COMPOUND QUERIES

Primitives can be combined into expressions with logical operators. An expression can take any of these forms, where *query* is a primitive or another expression:

query || *query*	*(logical OR)*
query && *query*	*(logical AND)*
! *query*	*(logical NOT)*
(*query* **)**	*(grouping to override precedence)*

OPERATOR PRECEDENCE

The precedence and associativity of the operators for attribute comparisons and formation of logical expressions are the same as in the C programming language:

highest precedence: **!**	*(right associative)*
lower precedence: **< <= > >=**	*(left associative)*
lower precedence: **== !=**	*(left associative)*
lower precedence: **&&**	*(left associative)*
lowest precedence: ||	*(left associative)*

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Display all versions of test **.c** for which the attribute **QAed** has the value Yes.

  % cat 'cleartool describe –s –ver /main'{QAed=="Yes"}' test.c'

- Attach the label **REL6** to the version of **test.c** that is already labeled **REL5**.

  % cleartool mklabel -ver '{lbtype(REL5)}' REL6 test.c
  ```
  Created label "REL6" on "test.c" version "/main/4".
  ```

- Attach an attribute to the latest version of **test.c** created since yesterday at 1 P.M. by user **asd**. Note the use of backslashes (\) to escape quote characters (") required to specify a string argument to **mkattr**.

  ```
  % mkattr -ver '{created_since(yesterday.13:00)&&created_by(asd)}' QAed \"No\" test.c
  Created attribute "QAed" on "test.c@@/main/5".
  ```

- List each branch named **rel2_bugfix** that occurs in an element to which a trigger named **mail_all** has been attached.

  ```
  % cleartool find . -branch 'brtype(rel2_bugfix)&&trtype(mail_all)' -print
  ./util.c@@/main/rel2_bugfix
  ```

SEE ALSO

config_spec, **pathnames_ccase**, **version_selector**

quit

Quits an interactive or Attache session

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command
MultiSite	multitool subcommand

SYNOPSIS

q·uit

DESCRIPTION

The **quit** command ends an interactive **cleartool** or **multitool** session or an Attache session, returning control to the parent process. In Attache, the **ws_helper** program exits as well. In ClearCase, ClearCase LT and MultiSite, you can also exit by typing a UNIX EOF character (typically CTRL+D) or entering the **exit** command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

EXAMPLES

- End a **cleartool** interactive session.

  ```
  cleartool> quit
  ```

- End a **multitool** interactive session with the **quit** synonym, **exit**.

  ```
  multitool> exit
  ```

- End a **cleartool** interactive session with the UNIX **EOF** character.

  ```
  cleartool> <CTRL+D>
  %
  ```

- End an Attache session and exit the **ws_helper** program.

  cmd-context **quit**

SEE ALSO

(Attache only) **attache_command_line_interface**, **attache_graphical_interface**

rebase

Changes the configuration of a UCM stream

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

- Begin a rebase operation using the graphical user interface:

  rebase –gr·aphical [**–vie·w** *rebase-view-tag*]

- Cancel or check the status of a rebase operation:

  rebase { **–can·cel** | **–sta·tus** [**–l·ong**] } [**–vie·w** *rebase-view-tag*]

- Preview a rebase operation:

  rebase –**pre·view** [**–s·hort** | **–l·ong**] [**–vie·w** *rebase-view-tag*]
  { **–rec·ommended** | { **–bas·eline** *baseline-selector* [,...] **–dba·seline** *baseline-selector* [,...] } }

- Begin a rebase operation:

  rebase
  { **–rec·ommended** | { **–bas·eline** *baseline-selector* [,...] **–dba·seline** *baseline-selector* [,...] } }
  [**–vie·w** *rebase-view-tag*] [**–com·plete**] [**–gm ·erge** | **–ok**] [**–q·uery** | **–abo·rt** | **–qal·l**]
  [**–ser·ial**] [**–f·orce**]

- Resume or complete a rebase operation:

  rebase { **–res·ume** | **–com·plete** } [**–vie·w** *rebase-view-tag*]
  [**–gm·erge** | **–ok**] [**–q·uery** | **–abo·rt** | **–qal·l**] [**–ser·ial**] [**–f·orce**]

DESCRIPTION

The **rebase** command reconfigures a stream by adding, dropping, or replacing one or more of the stream's foundation baselines. The file and directory versions selected by those new baselines (and thus their associated activities) then become visible in the stream's views.

Only labeled baselines can serve as foundation baselines.

Any changes made in the stream prior to a rebase operation are preserved during the rebase. For any file modified in the stream, **rebase** merges any changes that are present in versions of that

file in the new foundation baselines into the latest version of that file in the stream, thereby creating a new version. All such merged versions are captured in the change set of an integration activity that **rebase** creates. This integration activity becomes the view's current activity until the rebase operation is completed or canceled.

You must perform a rebase operation in a view belonging to the stream that is being rebased. Before starting the rebase operation, check in all files in that view. This way, you avoid potential problems caused by **rebase** merging changes into an already-checked out file—**rebase** cannot reliably unmerge those changes should you cancel the rebase operation.

As a rule, you should rebase development streams often to pick up changes in the project's recommended baselines. By doing so you can find integration problems early, when they are easier to fix. In addition, rebasing just before performing a deliver operation should reduce or eliminate the need for manual merging during the delivery.

Rules for Development Streams

A development stream can only be rebased to baselines that were created in its project's integration stream, or that serve as the integration stream's foundation baselines. This rule ensures that changes made in the development stream are based on the same line of development as the rest of its project's streams.

rebase is typically used to advance a stream's configuration; that is, to replace its current foundation baselines with more recent ones. However, you can also use **rebase** to:

- Revert to earlier baselines.
- Add baselines for components not currently in the stream's configuration.
- Drop components from the stream's configuration.

You cannot revert or drop a component that has been modified (that is, new versions have been created) in the development stream. Without this rule, **rebase** could potentially leave stranded the changes made against baselines that are no longer in the stream's configuration.

rebase allows different baselines to be moved in different directions—you can advance one baseline while reverting another.

Rules for Integration Streams

An integration stream can only be rebased to baselines created in other projects' integration streams (not development streams), or to import or initial baselines. See the **mkcomp** and **mkbl** reference pages for information about import and initial baselines.

Just as for development streams, **rebase** can advance or revert baselines in an integration stream's configuration, and add or drop components. It can also switch to another baseline that originates from a project different from the current foundation baseline; that is, a baseline that is neither an ancestor nor a descendant of the current foundation.

You cannot revert, switch, or drop baselines for components that are in the project's modifiable component list. This rule prevents **rebase** from leaving stranded the changes made to those components in the integration stream, as well as in the project's development streams.

PERMISSIONS AND LOCKS

Permissions Checking: None.

Locks: An error occurs if there are locks on any of the following objects: the UCM project VOB, the development stream.

Mastership: The current replica must master the development stream.

OPTIONS AND ARGUMENTS

INVOKING THE GRAPHICAL USER INTERFACE. *Default:* Command-line interface.

–gr·aphical
Invokes the graphical user interface for the rebase operation.

SPECIFYING THE REBASE VIEW. *Default:* The current working UCM view.

–vie·w *rebase-view_tag*
Specifies the UCM view in which to execute the **rebase** command. The view must be associated with a UCM stream that is the stream to be rebased.

CANCELLING A REBASE OPERATION.

–can·cel

Cancels a rebase operation and restores the stream's prior configuration. The option deletes the integration activity and any versions created by the rebase operation that are not yet checked in.

If any new versions have been checked in, the cancellation is halted and you are informed of completed merges and any checked in versions that resulted from the rebase activity. After undoing the merges and check-ins, you must issue the **rebase –cancel** command again to cancel the rebase operation.

OBTAINING THE STATUS OF A REBASE OPERATION.

–sta·tus
Displays the status of a rebase operation. You are informed whether a rebase operation is in progress in the specified stream; and if so, this option displays the new foundation baselines and the list of new activities being brought into the stream.

PREVIEWING THE RESULTS OF A REBASE OPERATION.

–pre·view
Shows what baselines would change and what new activities would be brought into the

stream if a rebase operation were to be executed in nonpreview mode. **–preview** fails if a rebase operation is in progress.

CONTROLLING OUTPUT VERBOSITY. *Default:* Varies according to the kind of output that the options described here modify: see the descriptions of **–status** and **–preview**.

–l·ong

As a modifier of **–status**, displays a list of activities and change sets, and a list of elements that will require merging, in addition to the default information displayed by **–status**.

As a modifier of –**preview**, displays a list of versions that potentially require merging, in addition to the default information displayed by –**preview**.

–**s·hort**
Modifies the –**preview** option. Displays only a list of the activities.

SPECIFYING BASELINES. *Default*: None.

–rec·ommended
Specifies that a development stream is to be rebased to its project's recommended baseline

–bas·eline *baseline-selector*[,...]
Specifies one or more baselines to use as new foundation baselines for the stream. See *Rules for Development Streams* and *Rules for Integration Streams* for criteria for specifying baselines.

baseline-selector is of the form: [**baseline:**]*baseline-name*[*@vob-selector*] and *vob* is the baseline's UCM project VOB.

–dba·seline *baseline-selector*[,...]
Specifies one or more baselines to remove from the stream's configuration. Files in those baseline's components are subsequently no longer visible or modifiable in the stream. See *Rules for Development Streams* and *Rules for Integration Streams* for criteria for specifying baselines.

baseline-selector is of the form: [**baseline:**]*baseline-name*[*@vob-selector*] and *vob* is the baseline's UCM project VOB.

RESUMING A REBASE OPERATION. *Default*: None.

–res·ume
Restarts a rebase operation from the point at which it has been suspended. A rebase operation can be interrupted with CTRL+C or when it encounters an external error or condition that requires more information. To continue the operation, reissue the rebase command with the –**resume** option. However, you cannot resume a rebase operation that has been successfully halted with the –**cancel** option.

COMPLETING A REBASE OPERATION. *Default*: None.

–com·plete
Completes a rebase operation. Checking in merged versions in the development view does not complete the rebase operation—you must use **–complete** to complete a rebase operation. You can use this option after a rebase has been suspended, for example, to resolve file conflicts. It resumes the command process, verifies that needed merges were done, checks in any versions that are checked out, and records changes in the change set for the rebase activity.

MERGE OPTIONS. *Default:* Works as automatically as possible, prompting you to make a choice in cases where two or more nonbase contributors differ from the base contributor. For general information, see the **findmerge** reference page.

-ok
Pauses for verification on each element to be merged, allowing you to process some elements and skip others. This option does not remain in effect after a rebase operation is interrupted.

–gm ·erge
Performs a graphical merge for each element that requires it. This option does not remain in effect after a rebase operation is interrupted.

–q·uery
Turns off automated merging for nontrivial merges and prompts you to proceed with every change in the from-versions. Changes in the to-version are automatically accepted unless a conflict exists. This option does not remain in effect after a rebase operation is interrupted.

–abo·rt
Cancels a merge if it is not completely automatic. This option does not remain in effect after a rebase operation is interrupted.

–qal·l
Turns off all automated merging. Prompts you to determine whether you want to proceed with each change. This option does not remain in effect after a rebase operation is interrupted.

–ser· ial
Reports differences with each line containing output from one contributor, instead of in a side-by-side format. This option does not remain in effect after a rebase operation is interrupted.

CONTROLLING COMMAND-LINE PROMPTS. *Default:* Prompt for user input.

–f·orce
Suppresses prompting for user input during the course of a rebase operation. The **–force**

option does not remain in effect if the rebase is interrupted: you must respecify it when you restart the rebase operation with –**resume** or –**complete**. The merge options to the rebase command are not affected by the –**force** option.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Start a rebase operation.

  cmd-context **rebase –recommended**

```
Advancing to baseline "BL1.119" of component "webo_modeler"
Updating rebase view's config spec...
Creating integration activity...
Setting integration activity...
Merging files...
No versions require merging in stream "chris_webo_dev".
Build and test are necessary to ensure that the merges were completed
correctly.
When build and test are confirmed, run "cleartool rebase -complete".
```

- Complete a rebase operation.

  cmd-context **rebase –complete**

```
Rebase in progress on stream "chris_webo_dev".
Started by "ktessier" at 06/06/00 15:36:42.
Merging files...
No versions require merging in stream "chris_webo_dev".
Checking in files...
Clearing integration activity...
Updating stream's configuration...
Cleaning up...
Rebase completed.
```

SEE ALSO

checkin, **checkout**, **deliver**, **findmerge**, **setactivity**

recoverview

Recovers a *dynamic view* database

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

- Recover files associated with deleted VOB or deleted directory:
  recoverview [–f·orce] { **–vob** *vob-identifier* | **–dir** *dir-identifier* }
  { **–tag** *view-tag* | *view-storage-dir-pname* }
- Synchronize a view with one or more VOBs, moving stranded objects to a known location:
  recoverview –syn·chronize [–vob *pname-in-vob* **]**
  { **–tag** *view-tag* | *view-storage-dir-pname* }

DESCRIPTION

The **recoverview** command repairs a view database and the associated private storage area of a dynamic view (a *snapshot view* has no private storage in the same sense as does a dynamic view). Typically, you use this command after a system crash or similar mishap. You may also want to use this command to regain access to stranded view-private files. (See the *RECOVERING VIEW-PRIVATE FILES: VIEW LOST+FOUND DIRECTORY* section.)

Automatic Recovery

When necessary, **recoverview** is invoked by a dynamic view's associated **view_server** process. Enter this command yourself if messages in the view log (**/var/adm/atria/log/view_log**) suggest view database corruption (for example, `INTERNAL VIEW DB ERROR`).

Possible Data Loss

recoverview uses **reformatview**;that is, recovery involves a dump/load of the view database. (See the **view** reference page.) **recoverview** deletes the old, invalid view database, which **reformatview** has renamed to **db.dumped**.

Depending on the state of the view database, this process may cause certain information to be lost. After a view is recovered, consult **/var/adm/atria/log/view_log** to investigate possible data loss. The **set-UID** bit is always lost on files that the view owner does not own. See the **reformatview** reference page for more information.

RECOVERING VIEW-PRIVATE FILES: VIEW LOST+FOUND DIRECTORY

NOTE: Snapshot views do not have **lost+found** directories.

A file in view-private storage is normally accessed through a VOB pathname. That is, the file appears to be located in the VOB, but is actually stored in the view. But this view-VOB correspondence can be disrupted:

- A VOB can become temporarily unavailable—for example, by being unmounted.
- A VOB can become permanently unavailable, by being deleted.
- A particular VOB directory can become unavailable permanently, by being deleted with an **rmelem** command.

In all these cases, view-private files that are accessed through the unavailable VOB structure become stranded; the files cannot be used for normal ClearCase operations, because there are no VOB pathnames through which they can be accessed. You can resynchronize your view with the available VOBs with the **–vob** and **–dir** options. This recovers stranded files by moving them into the view's lost-and-found area, located at *view-storage-dir-pname***/.s/lost+found.** Recovered files remain inaccessible to normal ClearCase operations; you can access them through the view storage directory, using standard operating system utilities and commands.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SYNCHRONIZING A VIEW WITH ONE OR MORE VOBS. The following option synchronizes the dynamic view with one or more VOBs. With this option, **recoverview** moves all stranded files to the dynamic view's lost and found subdirectory, **.s/lost+found**. A typical time to synchronize is after performing a relocate operation**.**

–syn·chronize [**–vob** *pname-in-vob*]
Synchronizes the view with all VOBs in which the view has created view-private files. With **–vob**, synchronizes the view only with the VOB specified by *pname-in-vob*.

FORCING RECOVERY. *Default:* **recoverview** displays a `Recovery not needed` warning message and exits immediately if the view database does not need to be recovered.

–f·orce
Performs a view database recovery, whether or not it's needed. Suppresses the warning message in the situation described above.

SPECIFYING THE VIEW. *Default:* None.

–tag *view-tag*
The view-tag of any registered dynamic view.

view-storage-dir-pname
The pathname of a dynamic view storage directory. Use the **lsview** command to list a view's storage directory.

CAUTION: Make sure that the current working directory is not the same as, or anywhere below *view-storage-dir-pname*.

RECOVERING VIEW-PRIVATE STORAGE. The following options take ClearCase-internal identifiers for a VOB or a VOB directory (*vob-identifier* and *dir-identifier*) as arguments. The **lsprivate** command uses these identifiers when listing an inaccessible VOB or VOB directory.

–vob *vob-identifier*
Moves all view-private files that correspond to the specified VOB to the dynamic view's **lost+found** directory.

–dir *dir-identifier*
Moves all view-private files that correspond to the specified directory element to the dynamic view's **lost+found** directory.

CAUTION: If the VOB or directory is still accessible, using these options is probably incorrect; it will *un*synchronize the view and VOB, not synchronize them.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

NOTE: recoverview writes status messages to the **view_log** file in **/var/adm/atria/log**; it does not print status messages on the standard output device.

- Synchronize the dynamic view **jackson_fix** with all VOBs in which it has created view-private files.

  cmd-context **recoverview –synchronize –tag jackson_fix**

- Synchronize a dynamic view whose storage directory is **/usr/home/jackson/ccviews/std.vws**. with the **/vobs/dvt** VOB.

cmd-context **recoverview –synchronize –vob /vobs/dvt **
/usr/home/jackson/ccviews/std.vws

- For dynamic view **cp_bugfix**, recover view-private files from a deleted VOB.

  cmd-context **lsprivate –tag cp_bugfix**

```
...
cleartool: Warning: VOB is unavailable -- using name:
"<Unavailable-VOB-1>".
  If it has been deleted use 'recoverview -vob <uuid>'
  VOB UUID is 1127d379.428211cd.b3fa.08:00:69:06:af:65
...
```

  cmd-context **recoverview –vob 1127d379.428211cd.b3fa.08:00:69:06:af:65 –tag cp_bugfix**

SEE ALSO

reformatview, **view**

reformatview

Updates the format of a view database

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

reformatview [–dump·p | –loa·d] { –tag *view-tag* | *view-storage-dir-pname* }

DESCRIPTION

The **reformatview** command changes the format of a *view database* from that used in a previous release of ClearCase or ClearCase LT to the current format. A view database is a set of binary files in the **db** subdirectory of the view storage directory. A new release may use a different database format to support new product features, to enhance storage efficiency, or to improve performance.

View database conversion involves two major steps:

- Dumping the existing database to a set of ASCII files. This step invalidates the view database, which is renamed to **db.dumped**. You cannot use the view until its database is reloaded.
- Loading the ASCII files into a new database that uses the new format.

NOTE: This does not overwrite the old, invalid view database; it remains in the view storage directory, as **db.dumped**, until you explicitly delete it with a standard operating system command.

A view's **view_server** process detects the need for reformatting and displays a message to this effect. **reformatview** itself writes status messages to **/var/adm/atria/log/view_log**, not to **stdout** or **stderr**.

You can also use **reformatview** to move a view storage area between hosts of different architectures—that is, hosts on which there are differences in the binary files that implement the view database.

Possible Data Loss

In the case of a *dynamic view*, if the view database requires recovery, some information may be lost in the dump/load process. For example, if a view-private file is owned by someone other than the owner of the view storage area, **reformatview** always strips its *set-UID* bit (if the bit is set). In addition, some view-private files may be moved into the view's **lost+found** directory. See the **recoverview** reference page for details.

In the case of a *snapshot view*, the lost information may included loaded files as well as view-private files (and snapshot views have no **lost+found** directories).

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

FORCING A DUMP. *Default:* If a view's database does not require reformatting (it is up to date), **reformatview** displays a message and takes no other action; if the database is out of date, **reformatview** performs a dump, then a load.

–dum·p
Performs only the first step—creating an ASCII dump of the view database in file **view_db.dump_file** in the view storage directory.

–loa·d
Performs only the second step—replacing the old view database with a new one, using the contents of a previously created ASCII dump file.

SPECIFYING THE VIEW. *Default:* None.

–tag *view-tag*
The view-tag of any registered view.

view-storage-dir-pname
The pathname of a view storage directory.

CAUTION: Make sure that the current working directory is not the same as, or anywhere below, *view-storage-dir-pname*.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Reformat a view whose view-tag is **jackson_old**.

cmd-context **reformatview –tag jackson_old**

- Reformat a view whose storage directory is **/usr/home/jackson/ccviews/fix.vwS**.
  cmd-context **reformatview /home/jackson/ccviews/fix.vws**

FILES

/var/adm/atria/log/view_log

SEE ALSO

errorlogs_ccase, **recoverview**

reformatvob

Updates the format of a VOB database.

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only:

  reformatvob [**–dum·p** | **–loa·d**] [**–rm**] [**–f·orce**] [**–to** *dumpfile-dir-pname*]
  [**–hos·t** *hostname* **–hpa·th** *local-pname* **–gpa·th** *global-pname*]
  vob-storage-dir-pname

- ClearCase LT only:

  reformatvob [**–dum·p** | **–loa·d**] [**–rm**] [**–f·orce**] [**–to** *dumpfile-dir-pname*]
  vob-storage-dir-pname

DESCRIPTION

NOTE: Always back up a VOB's storage directory before using this command.

reformatvob is a one-way command. The dump and load phases must be allowed to complete (although they can take place at different times). You cannot abort and undo a reformat operation after you have started it; you can only restart and complete the operation.

reformatvob changes the format of a VOB database from a format used in a previous release of ClearCase or ClearCase LT to the current format. A new release may use a different database format to support new product features, to enhance storage efficiency, or to improve performance.

reformatvob also performs the actions of the **checkvob –setup** command. This **checkvob** setup processing must be completed to use the **checkvob** command. If this processing is interrupted during the **reformatvob** command execution, you must run the **checkvob** command manually. (See the **checkvob** reference page for details.)

You can also use **reformatvob** for other purposes:

- In ClearCase and Attache, to move a VOB storage directory between hosts of different architectures, that is, hosts with different binary formats for the files that implement the VOB database
- In ClearCase, Attache, and ClearCase LT, to clean up a VOB database, physically deleting records that have been logically deleted by **vob_scrubber**

In both cases, the VOB database has the same internal format, and **checkvob –setup** is not invoked.

reformatvob locks the VOB before reformatting it. If the VOB is already locked, **reformatvob** proceeds with the reformatting and then unlocks the VOB.

NOTE: **reformatvob** does not overwrite the old, invalid VOB database; it renames the old database to **db.***date*. The old database remains in the VOB storage directory until you delete it with a standard operating system command.

Dumping and Loading

VOB database reformatting involves two phases:

- Dumping the existing VOB database to a set of ASCII files. This phase is performed by the **db_dumper** program. **reformatvob** evaluates disk-space availability before beginning its work, and displays a message if disk space is insufficient. See the *Working with Limited Disk Space* section.
- Loading the ASCII files into a new VOB database that uses the current format. This phase is performed by the **db_loader** program.

By default, both phases are performed at the same time. The **–dump** and **–load** options enable you to perform them separately.

Both **db_dumper** and **db_loader** are **setUID-root** programs (see **setuid(2)**). An invocation of **reformatvob** fails if these programs have the wrong ownership or the wrong access mode:

```
cleartool: Error: Database dumper "<VOB-pname>/db.reformat/db_dumper"
must be setUID and owned by the superuser.
```

See the **db_dumper** reference page for more information.

Registering the VOB (Again)

reformatvob updates (or creates, if necessary) the VOB's entry in the network's **vob_object** registry file. However, **reformatvob** does not affect the **vob_tag** registry file. If the reformatted VOB does not have an appropriate VOB-tag registry entry, **reformatvob** displays a message advising you to create one or more VOB-tags with **mktag**. For more information on VOB-tags and registries, see the **mkvob** and **registry_ccase** reference pages.

Working with Limited Disk Space

Running **reformatvob** can substantially increase the amount of disk space that the VOB storage directory uses:

- By default, the old VOB database is preserved in a renamed subdirectory of the VOB storage directory. You can use the **–rm** option to discard the old VOB database. Alternatively, you can use a standard operating system utility to discard it later—for example, after you have mounted the reformatted VOB and verified its accessibility.
- By default, the ASCII dump files are created within the VOB storage directory. You can use the **–to** option to create these files in another location.

Following are **reformatvob**'s disk-space requirements, based on the size of the existing VOB database as the unit:

Data Structure	Space Required	Need for Space Eliminated by
Existing VOB database	100%	**–rm**
ASCII dump files	100%	**–to**
Elbow room	10%	(Always required)

Thus, if you use neither **–rm** nor **–to**, **reformatvob**'s disk-space needs are approximately 210% of the size of the VOB database.

NOTE: If you use the **–to** option, you may see a warning that there is not enough disk space on the disk on which the VOB resides. The **reformatvob** command does not check the space in the location you specified with the **–to** option, so if you know that there is enough space, you can ignore this warning.

Restarting an Interrupted Reformat

There are no ill effects if a **reformatvob** command is interrupted (for example, by a system crash). Enter the command again to complete the reformatting. If **reformatvob** is interrupted after the dump phase completes, reentering the command starts with the load phase.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, **root**. See the **permissions** reference page.

Locks: No locks apply.

Other Restrictions: In ClearCase and Attache, the VOB storage directory must physically reside on the host where you enter this command. In ClearCase LT, you must enter this command at the ClearCase LT server host.

In all cases, The current working directory must not be at or below the VOB storage directory. Your shell must not have a view context—neither set view nor working directory view.

OPTIONS AND ARGUMENTS

PARTIAL REFORMAT. *Default:* Performs a complete reformat, including both the dump and load phases.

–dum·p
Performs only the first phase of the reformatting process—creating an ASCII dump of the current VOB database.

–loa·d
Performs only the second phase of the reformatting process—creating a new VOB database using a previously created ASCII dump.

PRESERVING A BACKUP OF THE VOB DATABASE. *Default:* The original VOB database directory (subdirectory **db** of the VOB storage directory) is preserved through renaming. During the dump phase, it is renamed to **db.reformat**; during the load phase, it is renamed again, to a name that includes a date stamp (for example, **db.02.18**).

–rm
Deletes the original VOB database during the load phase.

CONFIRMATION STEP. *Default:* Before beginning its work, **reformatvob** prompts you to confirm that you want to reformat the VOB database.

–f·orce
Suppresses the confirmation step.

ALTERNATE LOCATION FOR ASCII DUMP FILES. *Default:* The dump phase creates the ASCII dump files within the VOB storage directory.

–to *dumpfile-dir-pname*
(Do not use in conjunction with **–load**) Creates the ASCII dump files within the specified directory, which must not already exist.

VOB REGISTRY OPTIONS. *Default:* Using the *vob-storage-dir-pname* argument, **reformatvob** creates or updates the **vob_object** registry file; it leaves the **vob_tag** registry file unchanged. The following options update the VOB-tag entry.

–hos·t *hostname*
–hpa·th *local-pname*
–gpa·th *global-pname*
See the **mkstgloc** reference page for information on these options.

SPECIFYING THE VOB. *Default:* None.

vob-storage-dir-pname
The pathname of a VOB storage directory. If you use ClearCase or Attache, also refer to the descriptions of **–host**, **–hpath**, and **–gpath** in the **mkstgloc** reference page.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Reformat a VOB whose storage directory is **/home/jones/tut/tut.vbs**.

  NOTE: This example shows selected status messages only; **reformatvob** actually produces more verbose messages. See the **checkvob** reference page for the **setup** output.

  ```
  % cd
  ```

  ```
  cmd-context  reformatvob /home/jones/tut/tut.vbs
  Reformat versioned object base "/home/jones/tut/tut.vbs"? [no] y
  Dumping database...
  Dumper done.
  Dumped versioned object base "/home/jones/tut/tut.vbs".
  Loading database...
  Loader done.
  Loaded versioned object base "/home/jones/tut/tut.vbs".
  ```

SEE ALSO

checkvob, **db_dumper**, **lsvob**, **mktag**, **mkvob**, **mount**, **register**, **registry_ccase**, **vob**, **vob_scrubber**

register

Creates an entry in the VOB or view object registry.

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only—Register a view:

  reg·ister –vie·w [**–rep·lace**]
  [**–hos·t** *hostname* **–hpa·th** *host-storage-pname*]
  view-storage-pname

- ClearCase and Attache only—Register a VOB:

  reg·ister –vob [–**ucm·project**] [**–rep·lace**]
  [**–hos·t** *hostname* **–hpa·th** *host-storage-pname*]
  vob-storage-pname

- ClearCase LT only—Register a view:

  reg·ister –vie·w [**–rep·lace**] *view-storage-pname*

- ClearCase LT only—Register a VOB:

  register –vob [**–ucmproject**] [**–replace**] *vob-storage-pname*

DESCRIPTION

The **register** command creates or replaces an entry in VOB or view object registries. The registries enable clients to determine the physical storage locations of VOBs and views they access. Note that **register** has no effect on the VOB or view tag registries. You can also use **register** to update an existing registry entry, or to re-register a VOB or view that was temporarily removed from service with **unregister**.

Other Commands that Affect Registries

The **mkview** and **mkvob** commands add an entry to the appropriate registry; the **rmview** and **rmvob** commands remove registry entries. You can use the **unregister** command to remove an

existing entry. The **reformatvob** command updates a VOB's object registry entry (or creates one, if necessary).

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

VIEW/VOB SPECIFICATION. *Default:* None.

–vob
Registers a VOB storage directory.

–ucm·project
Marks a VOB as a UCM project VOB in the registry.

–vie·w
Registers a view storage directory.

OVERWRITING AN EXISTING ENTRY. *Default:* An error occurs if the view or VOB storage directory already has an entry in the registry.

–rep·lace
Replaces an existing registry entry. (No error occurs if there is no preexisting entry.)

SPECIFYING THE LOCATION OF THE STORAGE DIRECTORY. *Default:* None.

view-storage-pname
The path to the view storage; to determine the path, use **lsview**.

vob-storage-pname
The path to the VOB storage; to determine the path, use **lsvob**.

SPECIFYING NETWORK ACCESSIBILITY. *Default*: Values are derived from the *view-storage-pname* or *vob-storage-pname* arguments.

–hos·t *hostname*
–hpa·th *local-pname*
See the **mkstgloc** reference page for descriptions of how to use these options.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Register a VOB storage directory that was previously unregistered with the **unregister –vob** command.

  cmd-context **register –vob /vobstore/vob2.vbsfs**

- Register a view storage directory.

  cmd-context **register –view /viewstore/view3.vws**

- Replace the existing registry entry for a VOB storage directory, explicitly specifying the access path information.

  cmd-context **register –vob –replace –host corona –hpath /vobstg/tests.vbs \ –gpath /net/corona/vobstg/test.vbs /vobstg/test.vbs**

SEE ALSO

mktag, **mkview**, **mkvob**, **mount**, **umount**, **unregister**, **registry_ccase**

registry_ccase

Storage registry for VOBs and views

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure

SYNOPSIS

- Registry directory:

  /var/adm/atria/rgy

- Registry backup directory (on backup registry hosts only—ClearCase only)

  /var/adm/atria/rgy/backup

- VOB registry files:

  vob_object
  vob_tag

- View registry files:

  view_object
  view_tag

- Region registry file (meaningful in ClearCase only because ClearCase LT installations are limited to a single region):

  regions

- Storage path registry file:

  storage_path

- Request base registry files (ClearTrack sites only):

  bbase_object
  bbase_tag

- Site configuration registry file:

  site_config

- Configuration files:

rgy_hosts.conf
rgy_region.conf
rgy_svr.conf
vob_tag.sec

DESCRIPTION

Each ClearCase and ClearCase LT host in the network has a registry directory: subdirectory **rgy** of **/var/adm/atria**, the administration directory. On most hosts, the registry directory contains only the **rgy_hosts.conf** and **rgy_region.conf** configuration files. On one host in the network, the registry server host (in ClearCase LT, it is always the ClearCase LT server host), the registry directory contains information on all the VOBs and views in the local area network, organized into the following files:

vob_object	Registry of VOB storage directories
vob_tag	Registry of VOB-tags
view_object	Registry of view storage directories
view_tag	Registry of view-tags
regions	Registry of network region names (significant only for ClearCase installations)
storage_path	Registry of default VOB and view storage locations
site_config	Registry of local configuration parameters
bbase_object	(optional) Registry of ClearTrack request base storage directories
bbase_tag	(optional) Registry of ClearTrack request base tags

Never edit the registry files on the registry server host manually. Use the following **cleartool** subcommands to add, delete, or modify registry file entries:

Command Name	Registry Files Affected	Change
mktag -view	view_tag	Add or replace entry
mktag -vob	vob_tag	Add or replace entry
rmtag -view	view_tag	Delete entry
rmtag -vob	vob_tag	Delete entry
mkregion	regions	Add entry
mkreplica	vob_tag	Add entry
	vob_object	Add entry
mkstgloc	vob_tag	Add entry
	view_tag	Add entry
	regions	Add entry (ClearCase only)
mkview	view_tag	Add entry
	view_object	Add entry

Command Name	Registry Files Affected	Change
rmview	view_object	Delete entry
	view_tag	Delete entry
mkvob	vob_tag	Add entry
	vob_object	Add entry
rmregion	regions	Delete entry
rmreplica	vob_tag	Delete entry
	vob_object	Delete entry
rmstgloc	vob_tag	Delete entry
	view_tag	Delete entry
	regions	Delete entry (ClearCase only)
rmvob	vob_tag	Delete entry
	vob_object	Delete entry
register –view	view_object	Add or replace entry
register –vob	vob_object	Add or replace entry
setcache –view	site_config	Add or replace entry
setsite	site_config	Add or replace entry
unregister –view	view_object	Delete entry
unregister –vob	vob_object	Delete entry

The **lsview**, **lsvob**, **lsregion**, **lssite**, **getcache**, and **hostinfo –long** commands read and report information from the registry files. The **rgy_check** utility performs various registry file consistency checks.

CLEARCASE ONLY—REGISTRY SERVER

You designate one host in the local area network to be the registry server host. (In fact, there can be more than one, but VOBs and views cannot be shared between clients whose VOBs and views are registered on different registry server hosts.) The name of this host must appear in the file **/var/adm/atria/rgy/rgy_hosts.conf** on each host in the network. The ClearCase **albd_server** program running on the registry server host acts as the registry server process: it fields remote procedure call (RPC) requests for registry information from ClearCase client programs (and other server programs) around the network.

CLEARCASE ONLY—BACKUP REGISTRY SERVERS

You can designate one or more ClearCase hosts as backup registry server hosts. If the primary registry server fails, you can run **rgy_switchover** to activate a backup registry server and reset all client hosts accordingly.

A backup registry server host takes periodic snapshots of the primary registry host's registry files (see **rgy_backup**) and client list, and it stores these snapshot files in the directory **/var/adm/atria/rgy/backup**.

Each ClearCase host has a text file, **/var/adm/atria/rgy/rgy_hosts.conf**. The name of the primary registry server host appears on the first line, and the name of the backup registry server host appears on the second line. For example, the following **rgy_hosts.conf** file names **mercury** as the primary registry server host and **venus** as the backup registry server host:

```
% cat /var/adm/atria/rgy/rgy_hosts.conf
mercury
venus
```

See also **rgy_backup** and **rgy_switchover**.

CLEARCASE ONLY—NETWORK REGIONS

You can conceptually partition a local area network into multiple ClearCase *network regions*. Each region is a consistent naming domain: all hosts in the same region must be able to access all VOB storage directories and view storage directories using the same full pathnames. For example, all hosts in one network region might access a view storage directory on host **neptune** using this **automount(1M)**-style pathname:

```
/net/neptune/shared_views/rls3_clean.vws
```

Hosts in another network region may use a different name to access the same view storage directory:

```
/net/neptune_gw/shared_views/rls3_clean.vws
```

These hosts may be on a subnetwork that uses a different network interface to host **neptune**. You may need to place hosts that use a nonstandard auto-mount program, or that do not use an auto-mount program, in separate network regions.

Each host exists in exactly one network region and specifies that region in the file **/var/adm/atria/rgy/rgy_region.conf**.)The **hostinfo –long** command lists a host's network region.

Conceptually, each network region has its own *view-tag* registry and *VOB-tag* registry. However, the registry server host stores the only copies of the view-tag and VOB-tag files; each view-tag and VOB-tag entry includes a **–region** field, which assigns the tag to a particular region. It is common for a single VOB or view object to have multiple entries in the tag registry—one for each region in which it is registered.

NOTE: All VOBs and views that reside on a host must have tags in the host's network region.

Network regions are registered in the **regions** file in the **rgy** subdirectory on the registry host. Use the **mkregion**, **lsregion**, and **rmregion** commands to create, list, and remove regions.

FORMAT OF REGISTRY FILES

The following sections describe the fields in the various registry files.

vob_object

Each VOB storage directory in the network has one entry in the **vob_object** file. The entry is a single text line with these fields:

`-entry`	`vob_object`
`-hostname`	Host on which the VOB storage directory resides.
`-local_path`	Standard full pathname to the VOB storage directory on that host.
`-vob_replica`	Unique identifier (UUID) of the VOB. (For a replicated VOB, maintained by ClearCase MultiSite, this *VOB replica UUID* identifies the particular VOB replica at your site.)
`-vob_family`	*VOB family UUID*, which is shared by all replicas of the same VOB. Replicas are created and maintained by ClearCase MultiSite. Even if the VOB is not replicated, this UUID is different from the VOB replica UUID.

vob_tag

Each VOB storage directory in the network can have one VOB-tag per network region, and each VOB-tag has an entry in the **vob_tag** file. The entry is a single text line with these fields:

`-entry`	`vob_tag`
`-tag`	VOB-tag, which is a full pathname. A VOB's tag is the same as its mount point.
`-global_path`	Standard full pathname to the VOB storage directory that is valid on all hosts within the network region.
`-hostname`	Host on which the VOB storage directory resides.
`-mount_access`	Keyword: **private** or **public**.
`-mount_options`	System-dependent character string that records options to be invoked when you mount the VOB.
`-region`	Network region.
`-vob_replica`	Same as the like-named field in the **vob_object** file.
`-title`	(optional) Tag comment supplied with a **–tcomment** option on **mkvob** or **mktag**

view_object

Each view storage directory in the network can have one entry per network region in the **view_object** file. The entry is a single text line with these fields:

`-entry`	`view_object`
`-hostname`	Host on which the view storage directory resides.

`-local_path`	Standard full pathname to the view storage directory on that host.
`-view_uuid`	View's unique identifier (UUID).
`-owner`	User name of the view's creator

view_tag

Each view storage directory in the network can have one or more entries per network region (one per region is recommended) in the **view_tag** file. The entry is a single text line with these fields:

`-entry`	`view_tag`
`-tag`	View-tag, which takes the form of a simple file name. When a *dynamic view* is active on a host, its view-tag appears as an entry in the host's *view.root* directory (**/view**)
`-hostname`	Host on which the view storage directory resides.
`-global_path`	For all machines in this network region, the full pathname of the view storage directory.
`-region`	Network region.
`-view_uuid`	View's unique identifier (UUID).
`-title`	(optional) Tag comment supplied with a **–tcomment** option on **mkview** or **mktag**.

regions

Each network region has exactly one entry in the **regions** file. The entry is a single text line with these fields:

`-entry`	`regions`
`-tag`	Region name (character string <= 32 characters)
`-title`	(optional) Region comment; see **lsregion** for details.

storage_path

Each default location for a VOB or view storage directory has exactly one entry in the **storage_path** file. The entry is a single text line with these fields:

`-entry`	`storage_path`
`-type`	Type of storage location (`vob` or `view`).
`-hostname`	Host on which the storage location resides.
`-region`	Network region.
`-global_path`	The full pathname of the storage location, valid on all hosts within the network region.

bbase_object

(ClearTrack sites only) Each ClearTrack request base in the network can have one entry per network region in the **bbase_object** file. The entry is a single text line with these fields:

`-entry`	`bbase_object`
`-hostname`	Host on which the request base resides.
`-local_path`	Standard full pathname to the request base on the local host.

bbase_tag

(ClearTrack sites only) Each ClearTrack request base in the network can have one tag per network region, and each request base tag has an entry in the **bbase_tag** file. The entry is a single text line with these fields:

`-entry`	`bbase_tag`
`-tag`	Request base tag.
`-hostname`	Host on which the request base resides.
`-global_path`	For all machines in this network region, the full pathname of the request base.
`-region`	Network region.

site_config

Stores local configuration parameters. See **setsite** for these parameters. The entry for each parameter is a single text line with these fields:

`-entry`	`site_config`
`-name`	Name of the parameter (such as `view_cache_size`).
`-value`	Default value for the parameter.

rgy_hosts.conf

A list of primary and backup registry server hosts. This file must exist on every ClearCase host. The file's first line names the network's primary registry server host. An optional backup registry host appears on the second line.

rgy_svr.conf

On a primary registry host, this file contains the string **master**.

rgy_region.conf

A single word (32 characters maximum, no white space) specifying the host's network region. Each ClearCase host belongs to a single network region. All hosts in a network region access VOB and view storage directories using the same global (that is, network) pathnames.

vob_tag.sec

The network's *VOB-tag password*, an encrypted character string. You must specify this password to create a public VOB-tag with **mkvob** or **mktag**, or to remove a public VOB-tag with **rmtag**.

The **vob_tag.sec** file exists only on the registry server host. If it is empty or nonexistent, the *root* user must place a password in it using the **rgy_passwd** utility. For example:

```
% rlogin ccsvr05 -l root                    (log in to registry server host)
Password: <enter root password>
/usr/atria/etc/rgy_passwd                   (invoke encrypted-password utility)
Password: <enter VOB-tag password>
```

FILES

/var/adm/atria/rgy/vob_object
/var/adm/atria/rgy/view_object
/var/adm/atria/rgy/vob_tag
/var/adm/atria/rgy/view_tag
/var/adm/atria/rgy/rgy_hosts.conf
/var/adm/atria/rgy/rgy_region.conf
/var/adm/atria/rgy/rgy_svr.conf
/var/adm/atria/rgy/regions
/var/adm/atria/rgy/storage_path
/var/adm/atria/rgy/bbase_object
/var/adm/atria/rgy/bbase_tag
/var/adm/atria/client_list.db

SEE ALSO

albd_server, **hostinfo**, **lsclients**, **lsregion**, **lssite**, **lsstgloc**, **lsview**, **lsvob**, **mkregion**, **mkstgloc**, **mktag**, **mkview**, **mkvob**, **mount**, **register**, **rgy_backup**, **rgy_check**, **rgy_passwd**, **rgy_switchover**, **rmregion**, **rmstgloc**, **rmview**, **rmvob**, **setcache**, **setsite**, **umount**, **unregister**

relocate

Moves elements and directory trees from one VOB to another

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

relocate [**–f·orce**] [**–qal·l**] [**–log** *log-pname*] [**–upd·ate**]
pname [*pname ...*] *target-dir-pname*

DESCRIPTION

The **relocate** command moves elements, including directory trees, from one VOB to another. All related VOB database entries and data containers are moved to the target VOB. **relocate** preserves the "move from" VOB's namespace by substituting VOB symbolic links for moved elements.

NOTE: In Attache, after moving elements from one VOB to another, **relocate** does not move the corresponding elements in the workspace.

The more common use of **relocate** involves splitting a piece from one VOB and moving it to a newly created VOB. However, you can move an arbitrary collection of elements from one VOB to a location in any other VOB. You cannot use **relocate** to move an element to a new location in the same VOB. Use **cleartool mv** for this purpose.

For a *dynamic view*, view-private files and nonversioned DOs are not relocated. If a relocated directory contains view-private files, they are stranded; DOs are removed. See also *Relocating Derived Objects* and *Using recoverview –sync to Recover View-Private Objects and DO Data Files*.

WARNING: The **relocate** command makes irreversible changes to at least two VOBs and their event histories. We recommend that you not use it frivolously or routinely for minor adjustments. Furthermore, you are advised to stop VOB update activity before and during a **relocate** operation.

Overview of a Relocate Operation

Primary **relocate** processing phases:

- Select/checkout from source VOB
- Copy elements to target VOB
- Catalog elements in target VOB

- Update hyperlinks
- Remove elements from source VOB

When moving elements from one VOB to another, **relocate** does the following:

1. Locks all elements in the relocate set. (These locks are held until **relocate** removes the elements from the source VOB, or until you abort **relocate** and release them manually.)
2. Checks out source files and directories, and checks out the target directory.
3. Copies all elements into the target directory, creating a temporary, flat directory of OID-prefixed element names.
4. Copies metadata types to target VOB (label, attribute, element, attribute, trigger, and hyperlink type objects).

   Name collisions result in new type names of the form *type-name.n* (label type **REL3.1**, or branch type **r2_bugfix.1**, for example).
5. Converts flat target directory to directory tree by cataloging elements in their containing directories.
6. For borderline elements being relocated (these, by definition, are cataloged in at least one nonrelocated directory): replaces original catalog entries in source VOB directory versions with VOB symbolic links. Each link's text is a relative pathname. For example: **../../vobs/newhome**
7. Rebuilds event history for all elements.
8. Applies nonhyperlink metadata to target VOB elements (labels, attributes, and triggers).
9. Adjusts hyperlinks. Preserves all bidirectional hyperlinks, including **merge** links. (**relocate** does not preserve unidirectional hyperlinks whose targets are relocated.) See also **mkhlink**.

   Adds hyperlink of type **RelocationVOB** to the target VOB. (**catcr** uses such hyperlinks to resolve references to relocated objects.)
10. Removes elements from source VOB.
11. Checks in target directory, and checks in source directories from which elements were relocated.

Invoking relocate

Because **relocate** performs checkouts, creates new elements, and removes elements, invoke it from a view configured to perform these actions on the source and target VOBs. In particular, make sure your *config spec* can check out the elements to be relocated. For example, a view or config spec without a **CHECKEDOUT** rule is inappropriate for the purposes of **relocate**. See the **config_spec** reference page for more information on view configuration.

WARNING: Do not run multiple **relocate** commands in the same view in the same directory. Because **relocate** checks out and checks in files in the directory, simultaneous commands will

interfere with each other. To ensure no interference, we recommend that you run only one **relocate** command at a time in a VOB.

Perform a relocate operation in the same view, or with the same config spec, that you will use to adjust makefiles, rebuild libraries, reset config specs, modify development tools, or complete any other work that may accompany the relocation task. See also *AFTER RELOCATION*.

If you are using ClearCase MultiSite, see *Relocating Elements from Replicated VOBs*.

Selecting Elements to Relocate

The term *selection set* refers to the complete collection of file and directory elements implied by your *pname* arguments to **relocate**, irrespective of your view. For example, the command **relocate file1 dir2 /vobs/newhome** yields a selection set that includes **file1**, **dir2**, and all files and directories cataloged in all versions of **dir2** and its subdirectories. (Note that this means the selection set can include elements for which your current config spec does not select any version.) **relocate** scans the selection set and extracts a subset, the *relocate set*, which it moves to the target VOB.

Often, the selection set and relocate set are identical. By default, **relocate** scans the selection set and adds these elements to the relocate set:

- Each element visible in the current view
- Each element cataloged only by a directory in the selection set

In other words, **relocate** leaves behind any element that is not visible in the current view and is cataloged in a directory outside the selection set. For example, suppose you relocate directory element **lib2**, which catalogs file element **file.c** (that is, at least one version of **lib2** catalogs **file.c**). Your view does not select any version of **file.c**. Directory element **lib4**, which you are not relocating, does catalog **file.c**. By default, **file.c** stays behind.

The **–qall** option permits you to interactively confirm or override **relocate**'s default handling of each borderline element: it does not query on all elements in the selection set. A borderline element is one that is cataloged both in a directory being relocated and in a directory *not* being relocated. **relocate**'s default handling of a borderline element depends on whether the element is visible to your current view: a visible element (config spec selects a version) moves by default. An invisible element (config spec selects no version) does not.

Running relocate in Update Mode

Because **relocate** locks elements to be moved, relocating elements from a production VOB prevents users from working on them. If the elements to be moved must remain in use during the relocation, you can use **relocate**'s update mode (**relocate –update**). You can also use update mode to update the target VOB incrementally with changes that have occurred since **relocate** was last run. However, we recommend that you use update mode sparingly.

You cannot switch from using update mode to not using it. We recommend you not use update mode if you plan to leave some elements in the original VOB. (If you do use it in this situation, you must clean up the original elements and create symbolic links manually.)

CAUTION: If you use update mode, you must not make any changes to the relocated elements in the target VOB until you have stopped using the originals.

In update mode, **relocate** works as follows:

- Does not create symbolic links in the source VOB to replace existing catalogs; it leaves the catalogs in place
- Allows you to relocate the VOB root and **lost+found** directories
- Does not stop relocation when it encounters a nonlocally mastered element; it relocates the element
- Does not remove the elements from the source VOB
- Does not check out or modify the source directory
- Does not lock the elements in the relocate set

When you use update mode incrementally, **relocate** rescans all elements to update them with any changes. In regular mode, **relocate** determines where it left off and continues, assuming that elements it has already relocated do not need further processing. In update mode, **relocate** does not make this assumption: because update mode is used incrementally and no locks are put on the original elements, the elements may have changed since the last time **relocate –update** was run.

There are certain changes (usually deletions of objects) that cause **relocate** to fail when updating existing elements. If you perform **rmver**, **rmbranch**, or **rmelem** operations on the original elements, these changes are not reflected in the destination VOB. **relocate** creates objects; it does not remove them.

In addition, a deletion combined with a replacement creation can cause an error. For example, if a branch named **foo** is removed in the original VOB and a new branch named **foo** is created to replace it, **relocate** fails when it tries to update the destination element. Because **relocate** did not remove the old branch, it cannot create the new one. To work around this problem, remove the destination element; on the next run of **relocate**, the element is re-created.

Relocating Derived Objects

NOTE: Derived objects are created only in dynamic views.

You can relocate checked-in derived objects as you do any element. Configuration records (CRs) for these DOs move to the target VOB's database.

All nonversioned DOs (shared and view-private)—and their config records—are deleted when **relocate** removes their containing directory from the source VOB. That is, DO and config record objects are deleted from the VOB database (not moved) with the equivalent of **rmdo** operations. See also *Using recoverview –sync to Recover View-Private Objects and DO Data Files*.

Relocating Elements to an Existing VOB

When moving elements to a VOB that is not currently empty (new), be careful to avoid the following problems:

- Target VOB locks on branches, types, or elements (use **lock –replace –nuser** to work around this problem)
- Name collisions

Relocating Elements from Replicated VOBs

When relocating elements from one replica of a replicated VOB to another VOB, observe the following rules and guidelines:

- You must be an element's master to relocate it.
- You must replicate the *target VOB* at each site that includes the *source VOB*. Relocate elements from VOB1 replica to VOB2 at local site; then replicate VOB2 at each VOB1 replica site.

  After a relocate operation at site A, cross-VOB hyperlinks will "dangle" at site B's replica if either site B does not include a replica of the target VOB used at site A, or site B has not been updated by a **syncreplica** operation since the relocate operation at site A.
- You must coordinate the relocate operation with the administrators of all of the VOB's replicas. For example, if replica A is locked for **relocate**, and a checkin happens at replica B in the meantime, that checkin is lost to replica A.
- You may need to adjust mastership for metadata type objects. When **relocate** copies them to the target VOB, it creates them with local mastership.

Relocation and Event History

For each relocated element, two events record the relocate event:

- In the source VOB, a remove element event on the VOB object
- In the target VOB, a relocate event on the relocated element

Relocated element event histories are otherwise preserved nearly intact. (Some minor events are lost.)

Relocate Log File

By default, **relocate** creates a log file in the current directory with the name **relocate.log.***date-time*. You can use **–log** to send log output to another location.

Interrupting and Restarting relocate

In general, you can interrupt and restart **relocate** operation, if you take these precautions:

- Do not release locks that **relocate** sets in the source VOB.
- Do not modify elements in the selection set (check them in, apply labels to them, and so on).
- Record your answers to **–qall** queries the first time through.

If restarted with an identical command line, **relocate** can determine where to resume processing.

The final phase in a **relocate** operation, removing elements from the source VOB, is not restartable. The relocate log file records the OIDs of elements to be removed. If you interrupt **relocate** during this phase, you must use the logged OIDs to remove elements manually (**rmelem** accepts arguments of the form **oid:***oid*).

AFTER RELOCATION

Modifying Views/Config Specs to Find Relocated Elements

In some cases, config specs for existing views may require changes to find elements that have been relocated. In source VOB directory versions that catalog relocated elements, **relocate** replaces the original catalog entry for a relocated element with a VOB symbolic link. Each link's text is a relative pathname, **../../newhome**, for example . Views left to access elements via symbolic links may run into problems. For example, you cannot check out a VOB symbolic link, even if it points to an element. There are several ways around this limitation, providing a view with a more direct path to elements in the target VOB:

- Reset views to use pathnames to the new VOB.
- Add labels to relocated versions, and configure the view to select these elements with a label-based rule.
- Apply a particular branch type to relocated elements, and configure the view to select this branch.

Similarly, you will want to examine your build scripts, triggers, and other tools that may need adjustments to accommodate relocated elements.

NOTE: In the source VOB, **relocate** completes its operation by checking in the parent directories of all relocated elements. Unlike all preceding directory versions, this last checked-in directory version does not include symbolic links to relocated elements. Presumably, the view in which you ran **relocate** selects this version, and to this view, relocated elements appear simply

removed. Working in this view affords an opportunity to track tools, builds, and other potentially "broken" references to the relocated elements.

Fixing Incorrect Symbolic Links

After relocating some elements, the corresponding symbolic links are not always correct for all views in which they may appear. To help resolve such problems, the **HyperSlink** hyperlink type links symbolic links to their objects. Use **describe** to get information on the problematic symbolic link; the **HyperSlink** makes evident how to fix the link.

Using recoverview –sync to Recover View-Private Objects and DO Data Files

relocate does not move view-private objects when it relocates their containing directories. If you do not move or back up these objects, they are effectively lost. However, you can use **recoverview –sync** *view-tag* to move view-private files to the dynamic view's **lost+found** directory, under names like *OID*_**file3.c**.

relocate deletes all nonversioned DOs (both shared and view-private)—and their config records—when it relocates their containing directories. A stranded DO data file (its status as a VOB object is now gone, along with its corresponding VOB database entries) can be recovered by a view that still references it with **recoverview –sync** *view-tag*. This command moves all view-private files and referenced DOs to the dynamic view's **lost+found** directory.

Cataloging Relocated Elements in Multiple Versions of the Target Directory

relocate checks out the target VOB directory, relocates elements, and checks in the target directory. This means that relocated elements are cataloged only in the latest version of the target directory, giving these elements a sense of newness that may not be desirable. For example, if the target VOB has been in existence for some time, you may want previous target directory versions to catalog the relocated elements. In this case, you can use the **cleartool** or Attache **ln –nco** to add VOB symbolic links to non-LATEST directory versions—cataloging relocated elements in previous versions of *target-dir-pname*.

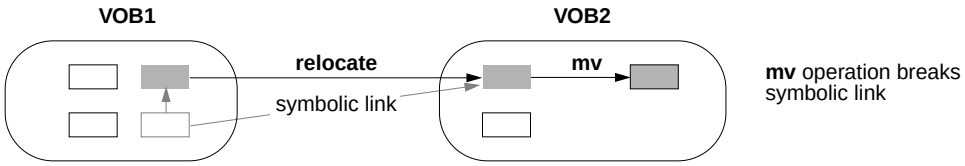
If you want relocated elements to be cataloged in the latest versions of *target-dir-pname* on multiple branches:

1. Set your config spec to select a branch of *target-dir-pname*, **/main/relocate_work**, for example.
2. Run **relocate**.
3. Perform directory **merge** operations from **/main/relocate_work** to other branches.

Moving a Relocated Element

If you move a relocated element with **mv**, a VOB symbolic link to the element from the source VOB is not updated. Instead, the link is left pointing at a nonexistent target.

Figure 13 Moving Relocated Element Does Not Update Symbolic Link from Source VOB



RESTRICTIONS

Permissions Checking: You must be the VOB owner (for both VOBs) or **root**to execute this command.

Locks: **relocate** first locks all elements to be moved from the source VOB. Then, it creates elements, element types, branch types, and so on in the destination VOB, as required to re-create all moved elements and their metadata. Therefore, **relocate** returns an error if any of the following objects are locked in the *source VOB*: VOB, element. It returns an error if any of the following objects are locked in the *destination VOB*: VOB, element, branch type, element type, label type, hyperlink type, attribute type. See the **permissions** reference page.

Other Restrictions: **relocate** cannot move checked-out elements. It fails during the selection phase if it finds any checked-out files among the ones it is going to move.

Also, **relocate** may fail if there are restrictive triggers on **checkout**, **checkin**, and **rmelem** commands. Because **relocate** runs these commands, triggers on these operations are also executed. If these triggers cause **relocate** to fail, you must disable the triggers or remove them from those operations, and run **relocate** again.

OPTIONS AND ARGUMENTS

SUPPRESSING THE CONFIRMATION QUERY. *Default:* After displaying the relocate set, **relocate** asks you to confirm that these are the elements you want to relocate.

–f·orce

Suppresses the confirmation step.

CONTROLLING SPECIAL CASE HANDLING. *Default:* **relocate** filters the selection set as described above in the subsection *Selecting Elements to Relocate*.

–qal·l

Prompts user to affirm or reject **relocate**'s handling of each borderline element—one that is cataloged both in a directory being relocated and in a directory not being relocated. The default answer in an individual case depends on the element's visibility in the current view: **yes** if the view selects some version of the element; **no** otherwise.

If you reject these defaults, the result is nonfunctional links that you must repair. If a version of an element is visible in the current view and you indicate it is not to be relocated, the result is a bad link in the target VOB. If a version of an element is not visible in the view and you indicate that it is to be relocated, the result is a bad link in the source VOB.

WRITING A LOG FILE. *Default:* **relocate** creates a log file in the current directory with the name **relocate.log.***date-time.*

–log *log-pname*
Creates a relocate log file at location *log-pname*.

RELOCATING IN UPDATE MODE. *Default:* **relocate** proceeds as described in *Overview of a Relocate Operation* on page 804.

–upd·ate
relocate runs in update mode, as described in *Running relocate in Update Mode* on page 806.

SPECIFYING WHICH FILES TO RELOCATE. *Default:* None. You must specify one or more elements to relocate, the *selection set*. **relocate** filters the selection set to construct a relocate set as described earlier in *Selecting Elements to Relocate*.

pname ...
Specifies the elements to be relocated. A *pname* can be a file element, directory element, or VOB symbolic link.

SPECIFYING A TARGET VOB AND DIRECTORY. *Default:* None. You must supply a target directory in a second VOB.

target-dir-pname
Specifies the directory in the target, or destination, VOB that will store the relocated elements. **relocate** checks out and modifies the version of this directory that is selected by your current view. The target directory must be in the same view as the source pathname (that is, you cannot specify a view-extended pathname for *target-dir-pname*).

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form

/vobs/*vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Move subdirectory **glib** (and its one file, **file.c**) from **/vobs/lib** to the newly created VOB **/vobs/gui**. Query on borderline elements. To illustrate how **relocate** replaces element names with symbolic links in the source VOB, the example uses a relative pathname to specify the target VOB.

After relocating **glib**, examine the `RelocationVOB` hyperlink added to **/vobs/gui**.

% cd /vobs/lib

cmd-context **setcs –default**

cmd-context **relocate -qall ./glib ../gui**

```
Logfile is "relocate.log.09-Apr-99.14:11:37".
Selected "glib".
Selected "glib/file.c".
Do you want to relocate these objects?  [no] yes
Checked out "." from version "/main/3".
Checked out "/vobs/gui" from version "/main/0".
Locking selected objects
Locked "glib"
Locked "glib/file.c"
Recreating selected objects
Created "glib"
updated branch "/main"
    updated version "/main/0"
        created version "/main/1"
        Created "glib/file.c"
    updated branch "/main"
        updated version "/main/0"
        created version "/main/1"
Cataloging new objects
```

```
    cataloged symbolic link "/vobs/lib/glib/.@@/main/2/glib" ->
"../gui/glib"
    cataloged symbolic link "/vobs/lib/glib/.@@/main/3/glib" ->
"../gui/glib"
    cataloged "/vobs/lib/.@@/main/CHECKEDOUT.32/glib"
    cataloged symbolic link "/vobs/lib/glib/.@@/main/1/file.c" ->
"../gui/glib/file.c"
    cataloged symbolic link "/vobs/lib/glib/.@@/main/2/file.c" ->
"../gui/glib/file.c"
    cataloged "/vobs/gui/glib@@/main/1/file.c"
Removing original objects
    removed "glib/file.c"
    removed "glib"
Checked in "/vobs/lib/." version "/main/4".
Checked in "/vobs/gui/." version "/main/1".
```

cmd-context **describe vob:/vobs/gui**

```
versioned object base "/vobs/gui"
  created 09-Apr-99.13:50:16 by CCase Admin (clearadm.sys@propane)
  "relocate target for former directory /vobs/lib/gui"
  VOB storage host:pathname "propane:/usr1/vobstore/gui.vbs"
  VOB storage global pathname "/net/propane/usr1/vobstore/gui.vbs"
  VOB ownership:
    owner clearadm
    group sys
  Hyperlinks:
    RelocationVOB@33@/vobs/gui vob:/vobs/gui -> vob:/vobs/lib/
```

FILES

relocate.log.*date-time*

SEE ALSO

ln, **mkvob**, **mv**

rename

Assigns a new name to an existing object.

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command
MultiSite	multitool subcommand

SYNOPSIS

rename [–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
{ *old-object-selector new-object-selector* | –**gen·erate** *old-object-selector* }

DESCRIPTION

NOTE: To move or change the name of a ClearCase or ClearCase LT file or directory element, use the **mv** command.

The **rename** command renames a ClearCase, ClearCase LT or MultiSite object—for example, a VOB storage pool, a replica, or a type object such as an activity type.

If you are renaming a pool, no data container in the pool is affected.

If you are renaming a replica, the name change is propagated to other replicas, through the standard synchronization mechanism. This command is valid only at the replica that masters the VOB-replica object being renamed.

If you are renaming a type object, all instances of the type object, throughout the VOB, are also renamed. If the type object is global, all local copies of the type object are renamed. For example, if you rename a branch type from **bugfix** to **rel1.3_fixes**, all existing **bugfix** branches are also renamed to **rel1.3_fixes**. (For more information about global type renaming, see *Administering ClearCase*.)

RESTRICTION: A VOB cannot contain a *branch type* and a *label type* with the same name.

NOTE: Do not use this command to rename an instance of a type, for example to rename a particular branch of a particular element. For that purpose, use **chtype**.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: replica creator (for renaming a replica only), object owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, object.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE OLD AND NEW NAMES. *Default:* None.

old-object-selector
new-object-selector
The name of an existing object and a new name for it. Specify *object-selector* in one of the following forms:

vob-selector	**vob:***pname-in-vob* *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted). It cannot be the pathname of the *VOB storage directory*.
attribute-type-selector	**attype:***type-name*[**@***vob-selector*]
branch-type-selector	**brtype:***type-name*[**@***vob-selector*]
element-type-selector	**eltype:***type-name*[**@***vob-selector*]
hyperlink-type-selector	**hltype:***type-name*[**@***vob-selector*]
label-type-selector	**lbtype:***type-name*[**@***vob-selector*]
trigger-type-selector	**trtype:***type-name*[**@***vob-selector*]
pool-selector	**pool:***pool-name*[**@***vob-selector*]
hlink-selector	**hlink:***hlink-id*[**@***vob-selector*]
oid-obj-selector	**oid:***object-oid*[**@***vob-selector*]

The following object selector is valid only if you use MultiSite:

replica-selector	**replica:***replica-name*[**@***vob-selector*]

The following object selectors apply to UCM:

activity-selector	**activity**:*actvity-name*[**@***vob-selector*]

baseline-selector	**baseline**:*baseline-name*[@*vob-selector*]
component-selector	**component**:*component-name*[@*vob-selector*]
folder-selector	**folder**:*folder-name*[@*vob-selector*]
project-selector	**project**:*project-name*[@*vob-selector]*
stream-selector	**stream**:*stream-name*[@*vob-selector*]

For more information about object selectors, see the **cleartool** reference page.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Rename one of the current VOB's pools from **c_pool** to **c_source_pool**.

  cmd-context **rename -c "make pool name clearer" pool:c_pool pool:c_source_pool**
  ```
  Renamed pool from "c_pool" to "c_source_pool".
  ```

- List existing pools in the current VOB. Then, rename pool **do1** to **do_staged**.

  cmd-context **lspool –short**
  ```
  c_source_pool
  cdft
  ddft
  do1
  my_ctpool
  sdft
  ```

  cmd-context **rename pool:do1 pool:do_staged**
  ```
  Renamed pool from "do1" to "do_staged".
  ```

- Rename a branch type from **rel2_bugfix** to **r2_maint**. First, show the version tree for **util.c** with the **lsvtree** command. Then rename the branch type, and show the version tree again.

  cmd-context **lsvtree –short util.c**
  ```
  util.c@@/main/1
  util.c@@/main/rel2_bugfix
  util.c@@/main/rel2_bugfix/1
  util.c@@/main/3
  ```

  cmd-context **rename brtype:rel2_bugfix brtype:r2_maint**
  ```
  Renamed type from "rel2_bugfix" to "r2_maint".
  ```

cmd-context **lsvtree –short util.c**
```
util.c@@/main/1
util.c@@/main/r2_maint
util.c@@/main/r2_maint/1
util.c@@/main/3
```

- Rename the element type of **msg.c** and **hello.c** from **text_file** to **source_file**. Use **grep(1)** to extract the element name/value from the output of the **describe** command. (Note warning about renaming a predefined type.)

  cmd-context **describe msg.c hello.c | grep 'element type'**
  ```
  element type: text_file
  element type: text_file
  ```

  cmd-context **rename eltype:text_file eltype:source_file**
  ```
  cleartool: Warning: Renaming a predefined object!
  Renamed type from "text_file" to "source_file".
  ```

  cmd-context **describe msg.c hello.c | grep 'element type'**
  ```
  element type: source_file
  element type: source_file
  ```

- Rename an attribute attached to a version of element msg.c from **TESTED** to **QAed**. Use **describe** to show the name/value association before and after the name change.

  cmd-context **describe –aattr TESTED msg.c**
  ```
  msg.c@@/main/3
   Attributes:
   TESTED = "TRUE"
  ```

  cmd-context **rename attype:TESTED attype:QAed**
  ```
  Renamed type from "TESTED" to "QAed".
  ```

  cmd-context **describe –aattr QAed msg.c**
  ```
  msg.c@@/main/3
   Attribut
   QAed = "TRUE"
  ```

- Rename replica **paris** to **paris_louvre**.

  cmd-context **rename replica:paris paris_louvre**
  ```
  Renamed replica "paris" to "paris_louvre".
  ```

SEE ALSO

chactivity, **chchkpt**, **chevent**, **chpool**, **chtype**, **describe**, **lspool**, **lstype**, **mkpool**, **mkreplica** (in the *ClearCase MultiSite Manual*), **rmpool**, **rmtype**, **chmod(1)**

reqmaster

Sets access controls for mastership requests or requests mastership of a branch

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
MultiSite	multitool subcommand

SYNOPSIS

- Display or set the ACL for mastership requests:

  reqmaster –acl [**–edi·t** | **–set** *pname* | **–get**] *vob-selector*

- Set access controls for the replica or for branches:

  reqmaster [**–c·omment** *comment* | **–cq·uery** | **–nc·omment**]
  { { **–enable** | **–dis·able** } *vob-selector*
  | { **–den·y** | **–allow** } **–inst·ances** *branch-type-selector ...*
  | { **–den·y** | **–allow** } *branch-pname ...*
  }

- Request mastership of a branch:

  reqmaster [**–c·omment** *comment* | **–cq·uery** | **–nc·omment**]
  [**–lis·t**] *branch-pname ...*

DESCRIPTION

This command has three forms: two forms to configure access controls for mastership requests and one form to request mastership of a branch from the replica that masters the branch. For more information, see *ClearCase MultiSite Manual*.

SETTING ACCESS CONTROLS

To allow requests for mastership, the MultiSite administrator must set access controls at each replica:

- Add developers to the replica's access control list (ACL). Use the **–acl** option with **–edit** or **–set** to edit the ACL.
- Enable replica-level access. By default, replica-level access is not enabled. To enable it, use the **–enable** option.

Also, the branch type must allow mastership requests for branches of that type, and the branch object must allow mastership requests for the branch. By default, type-level and branch-level access are enabled. You can enable replica-level access, but deny requests for mastership of specific branches or for mastership of all branches of a specific type. Even if replica-level access is enabled, the **reqmaster** command fails if requests for mastership are denied at the type level or object level. Use the **–deny** option to deny requests at the type and branch level.

REQUESTING MASTERSHIP OF A BRANCH

This form of the **reqmaster** command contacts a sibling replica and requests that the replica transfer mastership of a branch to the current replica. You can also use **reqmaster** to display information about whether a mastership request for the branch will succeed.

If you specify multiple branches and the request fails for one or more branches, **reqmaster** prints error messages for the failures and continues processing the other branches.

TROUBLESHOOTING

If the **reqmaster** command fails, the error message indicates whether the failure occurred at the current replica or the sibling replica.

If the **reqmaster** command fails with the message `can't get handle`, enter the command again. If it continues to fail, contact the administrator of the sibling replica.

When you request mastership of a branch, the **reqmaster** command may complete successfully, but the mastership is not transferred to your current replica. In this case, verify that the synchronization packet was sent from the sibling replica and that your current replica imported it successfully.

Errors that occur during the mastership request process, including errors occurring during the synchronization export, are written to the **msadm** log file. To view this log, use the **cleartool getlog** command or the ClearCase Administration Console (available only on Windows computers).

For more information on error messages from the **reqmaster** command, see *ClearCase MultiSite Manual*.

RESTRICTIONS

Restrictions for setting access controls:

Permissions:

- To set the ACL, you must be VOB owner, *root* user (UNIX), a member of the *ClearCase group* (Windows), or have write permission on the ACL.
- To enable mastership requests at the replica level, you must be VOB owner, *root* user (UNIX) or a member of the *ClearCase group* (Windows).

Anyone can display the ACL with **reqmaster –acl –get**. See the **permissions** reference page in the *ClearCase Reference Manual*.

Locks: No locks apply.

Mastership: The replica must be self-mastering. For you to allow or deny mastership requests for a branch, your current replica must master the branch.

Restrictions for requesting mastership of a branch:

Permissions: You must be on the replica's ACL.

Locks: The **reqmaster** command fails if the branch, branch type, or VOB is locked.

Other restrictions: The **reqmaster** command fails in any of the following cases:

- Mastership requests are denied at any of the following levels: replica, type object, object.
- There are checkouts on the branch (except for unreserved, nonmastered checkouts).
- You specify a branch associated with a UCM stream.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See the **comments** reference page. To edit a comment, use **chevent**.

–c·omment *comment* | **–cq·uery** | **–nc·omment**
Overrides the default with one of the MultiSite comment options.

DISPLAYING OR SETTING ACCESS CONTROLS. *Default:* None. You must specify access controls. Specifying **–acl** with no other option displays the ACL for the current replica in the VOB family specified by *vob-selector*.

–acl [–edi·t | **–set** *pname* | **–get]** *vob-selector*
By default or with **–get**, displays the ACL for the current replica in the VOB family specified by *vob-selector*. With **–edit**, opens the ACL for the current replica in the editor specified by (in order) the **WINEDITOR** (UNIX only), **VISUAL**, or **EDITOR** environment variable. With **–set**, uses the contents of *pname* to set the ACL for the current replica.

Specify *vob-selector* in the form **vob:***pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–enable *vob-selector*
Allows mastership requests to be made to the current replica in the VOB family specified by *vob-selector*.

–dis·able *vob-selector*
Denies all mastership requests made to the current replica in the VOB family specified by *vob-selector*.

{ **–deny** | **–allow** } **–inst·ances** *branch-type-selector ...*
Denies or allows requests for mastership of all branches of the specified type. Specify *branch-type-selector* in the form **brtype:***type-name*[**@***vob-selector*]

type-name	Name of the branch type
vob-selector	VOB specifier; can be omitted if the current working directory is within the VOB. Specify *vob-selector* in the form [**vob:**]*pname-in-vob* *pname-in-vob* — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

{ **–deny** | **–allow** } *branch-pname ...*
Denies or allows requests for mastership of the specified branch. Specify *branch-pname* in the form *file-pname***@@***branch*. For example:

```
foo.c@@/main/v3.8
header.h@@\main\v1\bugfix
```

REQUESTING MASTERSHIP OF A BRANCH. *Default:* Sends a request for mastership to the master replica of the branch.

–lis·t
Does not request the mastership change; instead, displays branch type, and master replica of the branch, and whether a request would succeed.

branch-pname
Branch whose mastership you are requesting. For example:

```
foo.c@@/main/v3.8
header.h@@\main\v1\bugfix
```

EXAMPLES

- Display the ACL for the current replica in the VOB family **/vobs/dev**, and then change it to give full access to **ccadmin** and permission to request mastership to **susank** and **johng**.

  multitool reqmaster –acl –get vob:/vobs/dev
  ```
  # Replica hosmer@/vobs/dev
  # Request for Mastership ACL:
  Everyone: Read
  ```

cat > /tmp/hosmer_aclfile

```
# Replica hosmer@/vobs/dev
# Request for Mastership ACL:
User:foobar.com/ccadmin Full
User:foobar.com/susank Change
User:foobar.com/johng Change
```

multitool reqmaster –acl –set /tmp/hosmer_aclfile vob:/vobs/dev

multitool reqmaster –acl –get vob:/vobs/dev

```
# Replica hosmer@/vobs/dev
# Request for Mastership ACL:
User:foobar.com/ccadmin Full
User:foobar.com/susank Change
User:foobar.com/johng Change
```

- Allow requests for mastership for all branches mastered by the current replica in VOB family **/vobs/test_dev**, except for branches of type **v2.6_beta**.

  multitool reqmaster –enable vob:/vobs/test_dev

  ```
  Requests for mastership enabled in the replica object for
  "vob:/vobs/test_dev"
  ```

  multitool reqmaster –deny –instances brtype:v2.6_beta

  ```
  Requests for mastership denied for all instances of "brtype:v2.6_beta"
  ```

- Allow requests for mastership for all branches mastered by the current replica, except for the branch **cmdsyn.m@@/main/v2.6_integ**.

  multitool reqmaster –enable vob:/vobs/dev

  ```
  Requests for mastership enabled in the replica object for "vob:/vobs/dev"
  ```

  multitool reqmaster –deny cmdsyn.m@@/main/v2.6_integ

  ```
  Requests for mastership denied for branch "cmdsyn.m@@/main/v2.6_integ"
  ```

- Deny requests for mastership for all branches mastered by the current replica.

  multitool reqmaster –disable vob:/vobs/doc

  ```
  Requests for mastership  disabled in the replica object for
  "vob:/vobs/doc"
  ```

- Request mastership of the branch **cmdsyn.m@@/main/v2.6_dev**.

  multitool reqmaster cmdsyn.m@@/main/v2.6_dev

- Display mastership information about the branches **include.h@@/main/integ** and **acc.c@@/main**.

multitool reqmaster –list include.h@@/main/integ acc.c@@/main

```
multitool: Error: The following errors will be encountered
multitool: Error: acc.c@@/main
Request Mastership remote "reqmaster" operation (host "neon") would fail:
the requested operation is denied.
```

SEE ALSO

chmaster

ClearCase MultiSite Manual

reserve

Converts an unreserved checkout to reserved

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

res·erve [–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–cact**] *pname* ...

DESCRIPTION

The **reserve** command changes the checkout status of a *checked-out version* of an element to *reserved*. A temporary `reserve checkout of version` event record is written to the VOB database.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch.

Other restrictions: The following conditions must all be true:

- There are no reserved checkouts of the branch.
- The latest version on the branch is the predecessor version of your checked-out version.
- (If you checked out the branch with **–nmaster**) The current VOB replica masters the branch.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE ELEMENTS. *Default:* None.

–cwo·rk
(UCM) Reserves each checked-out version in the change set of the current activity in your view.

pname ...
One or more pathnames, each of which specifies an element. The checkout in the current view is changed, unless you use a view-extended pathname to specify another view.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Change the checkout status of an element to reserved.

  cmd-context **reserve util.c**
  ```
  Changed checkout to reserved for "util.c" branch "/main".
  ```

- Verify that you are the only user with a checkout of a certain file, and then convert your checkout from unreserved to reserved.

  cmd-context **lscheckout util.c**
  ```
  14 Mar.13:48    drp     checkout version "util.c" from /main/3
  (unreserved)
   "experiment with algorithm for returning time"
  ```

  cmd-context **reserve util.c**
  ```
  Changed checkout to reserved for "util.c" branch "/main".
  ```

SEE ALSO

checkin, **checkout**, **lscheckout**, **uncheckout**, **unreserve**

rgy_backup

Copies registry files and client list from primary registry server host to backup registry server host

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

rgy_backup

DESCRIPTION

By default, the ClearCase scheduler runs **rgy_backup** periodically. See the **schedule** reference page for information on describing and changing scheduled jobs.

When it runs on a host that is not a backup registry host, **rgy_backup** checks the backup server configuration and exits. When it runs on a backup registry server host, **rgy_backup** takes two snapshots:

- ClearCase registry files, **/var/adm/atria/rgy/***, on the primary registry server host (see also **registry_ccase**)
- Primary registry host's client list, which is maintained by the registry server host's **albd_server**

rgy_backup stores these snapshot files in the directory **/var/adm/atria/rgy/backup** on the backup registry server host. **rgy_backup** removes files older than 96 hours.

rgy_backup names the snapshot file after the original file and appends a time stamp to the file name. **rgy_backup** also creates a symbolic link, with the same name as the original file, that points to the snapshot file. For example, for registry file **vob_tag**, **rgy_backup** creates in the **backup** directory:

- **vob_tag .17-Jul-99.18:30:15**
- A symbolic link named **vob_tag** that points to **vob_tag .17-Jul-99.18:30:15**

If the primary registry server fails, you can run **rgy_switchover** to activate the backup registry server and reset all client hosts accordingly. The backup server must be running the same release of ClearCase as that running on the primary server.

rgy_backup logs its snapshot activity in the **/var/adm/atria/log/rgy_backup_log** file.

Designating a Backup Registry Host

Each ClearCase host has a text file, **/var/adm/atria/rgy/rgy_hosts.conf**. The name of the primary registry server host appears on the first line, and the name of the backup registry server host appears on the second line. For example, the following **rgy_hosts.conf** file names **mercury** as the primary registry server host and **venus** as the backup registry server host:

```
% cat /var/adm/atria/rgy/rgy_hosts.conf
mercury
venus
```

Typically, you name a backup registry server host on each ClearCase host by supplying information to the **site_prep** utility when you install ClearCase.

To change the backup registry server host:

1. Change the backup server designated at the primary registry server host.
2. Stop and restart ClearCase.

The next time **rgy_backup** runs, the primary registry server host updates the name of the backup registry server for all its clients.

Do not designate a backup registry host that is unsuitable to serve as primary registry server host in an emergency.

If your site uses multiple ClearCase registries, you cannot configure one primary registry server as the backup server for a different registry.

PERMISSIONS AND LOCKS

Permissions Checking: You must have write permission to the directory **/var/adm/atria/rgy**.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

None.

EXAMPLES

- On a backup registry host, take a snapshot of the ClearCase registry files manually.

  rgy_backup

FILES

/var/adm/atria/rgy/*
/var/adm/atria/rgy/backup/*
/var/adm/atria/rgy/rgy_hosts.conf
/var/adm/atria/rgy/rgy_svr.conf

/var/adm/atria/log/rgy_backup_log
/var/adm/atria/client_list.db

SEE ALSO

lsclients, **registry_ccase**, **rgy_switchover**, **schedule**

rgy_check

Check registry files for inconsistencies

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

- ClearCase only:

  rgy_check { **–vie·ws** | **–vob·s** } ... [**–reg·ion** *region*] [**–sto·rage**]

- ClearCase LT only:

  rgy_check { **–vie·ws** | **–vob·s** } ... [**–hst·orage**]

DESCRIPTION

The **rgy_check** command examines the contents of ClearCase or ClearCase LT VOB and/or view registries, and reports any errors or inconsistencies to **stderr**.

Registry problems have various causes:

- Editing registry entries directly, as with **emacs** or **vi**.
- Improper administration procedures; for example, removing a VOB with **rm** rather than with **rmvob**
- Faulty upgrade procedures; for example, migrating a VOB to a new release that introduces a database schema change without reformatting the VOB (using **reformatvob**)
- Defects in older releases of ClearCase or ClearCase LT

If **rgy_check** finds errors or inconsistencies, it displays a line like the following at the end of its output:

```
Error: 21 total registry errors/inconsistencies detected.
```

For each problematic registry entry, **rgy_check** displays the registry entry and a warning or error message.

General Problems

rgy_check reports the following general problems:

- Duplicate entries in the registry
- Malformed entries in the registry

Registration Anomalies

rgy_check reports the following VOB or view registration anomalies:

- Objects with no UUID
- Two objects with same UUID
- Objects with no host name
- Objects with no local (server) pathname
- Two objects pointing to same *host-local-path*
- Tags with no UUIDs
- Tags with UUIDs that do not match any object (stranded tag)
- Tag registry entries with no tag

Region-Related Problems

Region-related problems are more likely to occur ClearCase than in ClearCase LT because ClearCase installations are not restricted to a single region. However, in either case, **rgy_check** may report these problems:

- Objects with no associated tags in any region (stranded object)
- Tags in regions that are not in the region registry
- Tags with no global pathname
- Two tags in one region pointing to same object UUID
- Duplicate tags in the same region
- Tags in one region with duplicate global pathnames

Storage-Related Problems

In ClearCase, if you specify the **–storage** option, **rgy_check** also reports these problems:

- View-tags that point to global paths with missing or incorrect **.view** files:
  - Missing **.view** file (usually a missing view)
  - **.view** file with invalid contents
  - **.view** file that contains an incorrect view UUID (that is, the UUID points to wrong view)
- VOB-tags that point to global paths with missing or incorrect **replica_uuid** files:
  - Missing **replica_uuid** file (usually a missing VOB)
  - **replica_uuid** file with invalid contents
  - **replica_uuid** file with an incorrect UUID (that is, the UUID points to wrong VOB)

In ClearCase LT, if you specify the **–storage** option, **rgy_check** reports the same kinds of problems that ClearCase reports when you use **–storage**, except that view and VOB objects (rather than tags) are checked.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE KIND OF REGISTRY ENTRIES TO DISPLAY. *Default:* None.

–vie·ws
Checks the contents of the view-tag and/or view-object registries.

–vob·s
Checks the contents of the vob-tag and/or vob-object registries.

SPECIFYING THE REGION. *Default:* All regions.

–reg·ion *region*
Specifies the network region for which registry entries are to be checked.

CHECKING STORAGE. *Default:* None.

–sto·rage
Checks for the existence of registered VOB and/or view storage directories. Given a storage directory's existence, **rgy_check** looks for basic storage configuration problems as well. Typically, registered storage pathnames for multiple network regions are not accessible from a single host. It is common practice to use **–region** to confine storage checks to the current host's network region.

–hst·orage
Checks for the existence of registered VOB and/or view storage directories on the ClearCase LT server host. Given a storage directory's existence, **rgy_check** looks for basic storage configuration problems as well. You must run **rgy_check** at the ClearCase LT server host when you use this option.

EXAMPLES

- Check the VOB registry for errors and anomalies.

  rgy_check -vobs
  ```
  No registry errors/inconsistencies detected.
  ```

- Check VOB and view registries in the **devel** region (which includes the local host). Include storage directory checks. In this example, **rgy_check** finds a tutorial VOB from which the user has removed the VOB's **replica_uuid** information.

rgy_check -vobs -views -region devel -storage

```
rgy_check: Error: The VOB storage at /net/io/alh/ccasetut/tut.vbs has no
replica_uuid file.
This tag:
-tag = "\alh_IO_hw"
-global_path = "/net/io/alh/ccasetut/tut.vbs"
-hostname = "io"
-mount_access = "private"
-mount_options = ""
-region = "devel"
-vob_replica = "7d7031db.6dfb11cf.a398.00:80:c8:81:fa:e0"

rgy_check: Error: 1 total registry errors/inconsistencies detected.
```

SEE ALSO

registry_ccase, *Administering ClearCase*

rgy_passwd

Creates or changes encrypted VOB-tag registry password

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

rgy_passwd [**–pas·sword** *tag-registry-password*]

DESCRIPTION

The **rgy_passwd** command places an encrypted password in the *VOB-tag password file*: **/var/adm/atria/rgy/vob_tag.sec** on the network's *registry server host*. This file need not already exist.

rgy_passwd Knowledge of this password enables an administrator to create *public* VOBs. Nonprivileged users can mount public VOBs by using the ClearCase **mount** command. See the **mkvob**, **mktag**, and **mount** reference pages for more information on public VOBs.

Security Restrictions

- You must execute **rgy_passwd** on the registry server host.
- If the **vob_tag.sec** file already exists, you must be the owner of that file.
- The registry directory, **/var/adm/atria/rgy**, is protected so that only the *root* user can create the **vob_tag.sec** file.
- **rgy_passwd** maintains the access mode of the **vob_tag.sec** file at 400. You need not use **chmod(1)** before or after entering **rgy_passwd**.

OPTIONS AND ARGUMENTS

By default, **rgy_passwd** prompts you to type the new password.

–pas·sword *tag-registry-password*
Specifies the password on the command line.

CAUTION: This is a potential security breach because the password remains visible in your transcript pad.

FILES

/var/adm/atria/rgy/vob_tag.sec

DIAGNOSTICS

`rgy_passwd: Error: Not a registry server.`

This command must be executed on the network's registry server host.

`rgy_passwd: Error: Unable to open file "/var/adm/atria/rgy/vob_tag.sec": Permission denied.`

A **vob_tag.sec** file already exists, and you are not its owner.

EXAMPLE

- Create a VOB-tag registry password interactively.

  rgy_passwd
  `Password:` *<enter VOB-tag password>*

SEE ALSO

mktag, **mkvob**, **mount**, **registry_ccase**

rgy_switchover

Makes a backup registry server host the primary registry server host

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

rgy_switchover [**–time** *file-timestamp*]
[**–backup** *new-backup-rgy-host*] *old-rgy-host new-rgy-host*

DESCRIPTION

The **rgy_switchover** command upgrades a backup registry server host (see **rgy_backup**) to primary registry server host and resets ClearCase clients to use the new primary registry server host.

rgy_switchover *old-rgy-host new-rgy-host* does the following:

1. On *new-rgy-host*, copies the **/var/adm/atria/rgy/backup** directory to **/var/adm/atria/rgy**.
2. On *new-rgy-host*, notifies the **albd_server** process that it is now running on a registry server host.
3. Uses the client list snapshot to inform clients of the change.
4. Reconfigures each client to recognize the new primary registry server host.
5. Reports an error for any client it cannot update. Update these clients manually when they become accessible. Update a client by editing its **rgy_hosts.conf** file.

   If a host is down and misses the switchover reconfiguration, it operates in degraded mode when it comes back until you or the host's owner updates it manually. In degraded mode, a client tries to access the primary registry server host, and when that fails, it tries to access the backup registry server host (which is now the primary registry server host to clients successfully reconfigured by **rgy_switchover**).

rgy_switchover logs its activities to **/var/adm/atria/log/albd_log**.

NOTE: You must ensure that only one registry server is active at any time. When the failed primary registry server host returns, or if it is still running when you invoke **rgy_switchover**, login on *old-rgy-host* and revoke its status as primary registry server host:

1. Stop ClearCase.
2. Edit the **rgy_svr.conf** file to remove the string **master**.

3. Edit the **rgy_hosts.conf** file to specify the new registry server host.
4. Restart ClearCase.

SAMPLE SWITCHOVER PROCEDURE

In this sample procedure, primary registry server host **rgy1** fails, and the ClearCase administrator makes backup registry server host **rgy2** the new primary registry server host. This assumes **rgy_backup** has been executing successfully on **rgy2** . While **rgy1** is down and **rgy2** is the primary registry server host, **rgy3** becomes the backup registry server host. Later, **rgy1** becomes available again and the administrator reverts to **rgy1** as the primary registry server host.

1. Make **rgy2** the primary registry server host. Make **rgy3** the new backup registry server host. Make **rgy1** a provisional backup registry server host, so that when it returns to life, it takes a registry snapshot in preparation for returning to its role as primary registry server host:

   rgy_switchover –backup "rgy3 rgy1" rgy1 rgy2

2. Record the names of any client hosts for which the switchover fails. Reset these hosts by hand when they become available (or have their owners do so). Note that ClearCase hosts with client-only installations and ClearCase Attache clients cannot be reconfigured automatically and always appear on the returned list of "unreachable" clients.

   When a failed client becomes available, set the first line of its **rgy_hosts.conf** file to **rgy2**.

3. Host **rgy1** becomes available again.
4. **rgy1** is still configured as a primary registry server host. Reconfigure it to recognize **rgy2** as the primary registry server host:
   a. Log in to **rgy1**.
   b. Stop ClearCase.
   c. Edit **rgy_svr.conf** and remove the string **master**.
   d. Edit **rgy_hosts.conf** by changing its first line to **rgy2**.
   e. Restart ClearCase.
5. Run **rgy_backup** manually on **rgy1**, forcing it to take a snapshot of the active registry files on **rgy2** in preparation for returning **rgy1** to service as the primary registry server host:

   rgy_backup

   NOTE: Running **rgy_backup** does not cause a snapshot operation unless **rgy1** is configured as a backup registry server host on the primary registry server host (**rgy2**). **rgy1** is correctly configured because it was named in the **–backup** argument in Step #1.

6. Stop registry service on **rgy2**:
   a. Log in to **rgy2**.
   b. Stop ClearCase.

c. Edit **rgy_svr.conf** and remove the string **master**.

d. Edit **rgy_hosts.conf** by changing its first line to **rgy1**.

e. Restart ClearCase.

7. Make **rgy1** the primary registry server host. Return **rgy2** to its former role as the backup registry host:

   rgy_switchover –backup "rgy2" rgy2 rgy1

PERMISSIONS AND LOCKS

Permissions Checking: You must be the *root* user to execute this command. See the **permissions** reference page.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE NEW BACKUP REGISTRY SERVER. *Default:* **rgy_switchover** does not configure a new backup registry host.

–backup *new-backup-host-list*
: Configures the hosts named in *new-backup-host-list* as backup registry server hosts, after switching the current backup registry server host to primary registry server host. If you specify multiple backup registry server hosts, enclose the space-separated host names in quotes, like this: **"venus mars"**

SPECIFYING A TIME STAMP. *Default:* **rgy_switchover** uses the most recent registry backup files in the *new-primary-rgy-host*'s **/var/adm/atria/rgy/backup** directory.

–time *file-timestamp*
: Activates an alternate set of backup registry files. The *file-timestamp* much match an existing set of time-stamped files in **/var/adm/atria/rgy/backup**. By default, the ClearCase scheduler runs **rgy_backup** periodically and deletes backed-up registry files more than three days old.

SPECIFYING THE OLD AND NEW PRIMARY REGISTRY SERVERS. *Default:* None. You must specify the current and target primary registry server hosts.

old-rgy-host
: The current primary registry server host.

new-rgy-host
: The current *backup* registry server host that will become the new primary registry server host.

EXAMPLES

- Make backup registry host **beta** the new primary registry host.

rgy_switchover alpha beta

- Same as previous example, but make **omega** the new backup registry host.

  rgy_switchover –backup omega alpha beta

- Same as previous example, but add alpha to the backup host list. This approach is recommended when the primary registry host (**alpha**) failed, prompting the switchover, but you plan to return it to primary registry host status when it becomes available again.

  rgy_switchover –backup "omega alpha" alpha beta

FILES

/var/adm/atria/rgy/*
/var/adm/atria/rgy/backup/*
/var/adm/atria/rgy/rgy_hosts.conf
/var/adm/atria/rgy/rgy_svr.conf
/var/adm/atria/log/albd_log
/var/adm/atria/client_list.db

SEE ALSO

lsclients, **registry_ccase**, **rgy_backup**, **rgy_passwd**, **schedule**

rmactivity

Deletes a UCM activity

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

rmact·ivity [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–nc·omment**]
[**–f·orce**] *activity-selector ...*

DESCRIPTION

The **rmactivity** command deletes one or more UCM activities. The following restrictions apply:

- The activity can have no versions in its change set.
- The activity cannot be set as the current activity for a view.

If versions exist in the change set, you can delete the versions or move the versions to another change set with **chactivity –fcset –tcset**.

ClearQuest-enabled Projects

When executed in a view that is associated with a ClearQuest-enabled project, this command unlinks the activity from its associated ClearQuest record and deletes the activity but it does not delete the ClearQuest record.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the owner of the activity, the VOB owner, or **root**.

Locks: An error occurs if there is a lock on any of the following objects: the UCM project VOB or the activity.

Mastership: The current replica must master the activity.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the specifed activity is to be deleted.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE ACTIVITY. *Default:* None.

activity-selector ...
Specifies one or more activities to delete.

You can specify an activity as a simple name or as an object selector of the form [**activity**]**:***name***@***vob-selector*, where *vob-selector* specifies a project VOB (see the **cleartool** reference page). If you specify a simple name and the current directory is not a project VOB, then this command assumes the activity resides in the project VOB associated with the stream attached to the current view. If the current directory is a project VOB, then that project VOB is the context for identifying the activity.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove an activity that is set as the current activity in a view.
  a. Issue an **rmactivity** command. The error message tells you that the specified activity is in use by the view **java_parser_int**:

```
cmd-context rmactivity -f new_object_tree@/usr1/tmp/foo_project
cleartool: Error: Activity
"activity:new_object_tree@/usr1/tmp/foo_project" is setworked in view
"java_parser_int".

cleartool: Error: Unable to remove activity
"new_object_tree@/usr1/tmp/foo_project".
```

  b. Go to the view in which the activity is set and unset it:

```
cmd-context setact –none
Cleared current activity from view java_parser_int.
```

  c. Reissue the **rmactivity** command:

```
cmd-context rmactivity -f new_object_tree@/usr1/tmp/foo_project
Removed activity "new_object_tree@/usr1/tmp/foo_project".
```

SEE ALSO

chactivity, **lsactivity**, **mkactivity**, **setactivity**

rmattr

Removes an attribute from an object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmattr [–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
{ [**–ver·sion** *version-selector*] [**–pna·me**]
attribute-type-selector pname ...
| *attribute-type-selector object-selector ...* }

DESCRIPTION

The **rmattr** command removes one or more *attributes* from *VOB-database* objects. Attributes can be attached to objects by the **mkattr** command and by *triggers* (**mktrtype** –**mkattr**). See the **mkattr** reference page for a list of objects to which attributes can be attached.

rmattr deletes an *instance* of an attribute type object. To delete the attribute type object itself or to delete the type object and all its instances, use the **rmtype** command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, object group member, object owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, object type, object, attribute type.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE ATTRIBUTE TO BE REMOVED. *Default:* None.

attribute-type-selector

An existing attribute type. Specify *attribute-type-selector* in the form [**attype:**]*type-name*[**@***vob-selector*]

type-name	Name of the attribute type
vob-selector	Object-selector for a VOB, in the form [**vob:**]*pname-in-vob*. The *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted).

SPECIFYING AN OBJECT. *Default:* None.

pname ...

One or more pathnames, indicating file-system objects from which attributes are to be removed. If you don't use the **–version** option:

- A standard or view-extended pathname to an element specifies the version in the view.
- A VOB-extended pathname specifies an element, branch, or version—independent of view.

See the **mkattr** reference page for examples of *pname* arguments.

–pna·me

Indicates that *pname* is a pathname. You must use this option if *pname* has the form of an object selector.

–ver·sion *version-selector*

Specifies the version from which the attribute is to be removed. See the **version_selector** reference page for syntax details.

object-selector ...

One or more names of non-file-system objects from which attributes are to be removed. Specify *object-selector* in one of the following forms:

vob-selector	**vob:***pname-in-vob* *pname-in-vob* can be the pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted). It cannot be the pathname of the *VOB storage directory*.
attribute-type-selector	**attype:***type-name*[**@***vob-selector*]
branch-type-selector	**brtype:***type-name*[**@***vob-selector*]
element-type-selector	**eltype:***type-name*[**@***vob-selector*]
hyperlink-type-selector	**hltype:***type-name*[**@***vob-selector*]

label-type-selector	**lbtype:***type-name*[**@***vob-selector*]
trigger-type-selector	**trtype:***type-name*[**@***vob-selector*]
pool-selector	**pool:***pool-name*[**@***vob-selector*]
hlink-selector	**hlink:***hlink-id*[**@***vob-selector*]
oid-obj-selector	**oid:***object-oid*[**@***vob-selector*]

The following object selector is valid only if you use MultiSite:

replica-selector	**replica:***replica-name*[**@***vob-selector*]

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove the **Confidence_Level** attribute from the version of **msg.c** in the view.

  cmd-context **rmattr Confidence_Level msg.c**
  ```
  Removed attribute "Confidence_Level" from "msg.c@@/main/1".
  ```

- Remove the attribute **TESTED** from the most recent version of **hello.h** on the **main** branch that has the attribute value `"FALSE"`.

  cmd-context **rmattr –version '/main/{TESTED=="FALSE"}' TESTED hello.h**
  ```
  Removed attribute "TESTED" from "hello.h@@/main/2".
  ```

- Remove the **Responsible** attribute from the **main** branch of **hello.c**.

  cmd-context **rmattr Responsible hello.c@@/main**
  ```
  Removed attribute "Responsible" from "hello.c@@/main".
  ```

- Remove the **Author** attribute from a hyperlink of type **DesignDoc**.

  cmd-context **rmattr Author hlink:DesignDoc@393@/usr/hw**
  ```
  Removed attribute "Author" from "DesignDoc@393@/usr/hw".
  ```

SEE ALSO

lstype, **mkattr**, **mkattype**, **rename**, **rmtype**

rmbl

Removes a UCM baseline

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

rmbl [**–c**·**omment** *comment* | **–cfi**·**le** *pname* | **–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
[**–f**·**orce**] *baseline-selector ...*

DESCRIPTION

The **rmbl** command deletes one or more UCM baselines. Versions associated with the baseline are not deleted, only the baseline relationship among the versions. The following restrictions apply:

- The baseline cannot serve as a foundation baseline for any stream.
- The baseline cannot be an initial baseline for a component.
- The baseline cannot be deleted if it is a full baseline and serves as the backstop for any incremental baseline.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the owner of the baseline, the VOB owner, **root**.

Locks: An error occurs if there are locks on any of the following objects: the UCM project VOB, the baseline.

Mastership: The current replica must master the baseline.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**
Overrides the default with the option you specify. See the **comments** reference page.

The comment is stored in a deletion event on the VOB object.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the specifed baselevel is to be deleted.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE BASELINE. *Default*: None.

baseline-selector ...
Specifies one or more baselines to delete.

baseline-selector is of the form: [**baseline:**]*baseline-name*[*@vob-selector*] and *vob* is the baseline's UCM project VOB.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove a baseline.

  cmd-context **rmbl -f START.109@/usr1/tmp/foo_project**
  ```
  Removed baseline "START.109@/usr1/tmp/foo_project".
  ```

SEE ALSO

diffbl, **lsbl**, **mkbl**

rmbranch

Removes a branch from the version tree of an element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmbranch [–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery**
| **–cqe·ach** | **–nc·omment**]
[**–f·orce**] *pname ...*

DESCRIPTION

This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases.

The **rmbranch** command deletes one or more *branches* from their *elements*. For each branch, deletion entails the following:

- Removal from the entire branch structure from the *VOB database*: branch object and version objects
- Removal of all *metadata* items (labels, attributes, hyperlinks, and triggers) that were attached to the deleted objects
- Removal of all event records for the deleted objects
- (File elements only) Removal of the data containers that hold the deleted versions' file-system data
- Creation of a `destroy sub-branch` event record for the parent branch of the deleted branch

NOTE: If all of an element's versions are stored in a single data container, the deleted versions are removed logically, not physically.

To delete all instances of a branch and the branch type object, use the **rmtype** command.

Restrictions

You cannot delete these branches:

- A branch that is checked out
- An element's **main** branch
- A branch from which someone has checked out elements (see the reference page for **uncheckout**)

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: branch creator, element owner, VOB owner, or **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, pool (nondirectory elements only).

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

CONFIRMATION STEP. *Default:* **rmbranch** prompts for confirmation before deleting anything.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE BRANCHES TO BE REMOVED. *Default:* None.

pname ...
One or more VOB-extended pathnames, indicating the branches to be deleted. Examples:

```
foo.c@@/main/bugfix
/vobs/proj/include/proj.h@@/main/temp_482
```

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form

/vobs/*vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Delete the **maintenance** branch of element **util.c**.

cmd-context **rmbranch util.c@@/main/maintenance**

```
Branch "util.c@@/main/maintenance" has 0 sub-branches, 2 sub-versions
Remove branch, all its sub-branches and sub-versions? [no] yes
Removed branch "util.c@@/main/maintenance".
```

- Verify, with the **lsvtree** command, that element **msg.c** has a **patch2** branch. Then, delete that branch without prompting for confirmation.

  cmd-context **lsvtree –branch /main/patch2 msg.c**

  ```
  msg.c@@/main/patch2
  msg.c@@/main/patch2/1
  ```

  cmd-context **rmbranch –force msg.c@@/main/patch2**

  ```
  Removed branch "msg.c@@/main/patch2".
  ```

SEE ALSO

lsvtree, **mkbranch**, **mkbrtype**, **rmtype**, **rmver**

rmcomp

Removes a UCM component

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

rmcomp [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment] [–f·orce]** *component-selector ...*

DESCRIPTION

The **rmcomp** command deletes a UCM component object. Elements of the component and the VOB associated with the component are not deleted. The following restrictions apply:

- There cannot be any baselines of the component other than the initial baseline
- The component's initial baseline cannot be in use as a foundation baseline for a stream.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the owner of the component, the VOB owner, or **root**.

Locks: An error occurs if there are locks on any of the following objects: component, UCM project VOB.

Mastership: The current replica must master the component.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the specifed component is to be deleted.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE COMPONENT TO BE DELETED. *Default*: None.

component-selector ...

Specifies one or more components to delete

component-selector is of the form: [**component:**]*component-name*[*@vob-selector*] and *vob* is the component's UCM project VOB.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove a component that contains baselines.

  a. Issue the **rmcomp** command for a specified component:

```
cmd-context  rmcomp parser@/usr1/tmp/foo_project
Remove component "parser@/usr1/tmp/foo_project"?  [no] yes

cleartool: Error: Cannot remove component that has baselines other than
the initial baseline.

cleartool: Error: Unable to remove component
"parser@/usr1/tmp/foo_project".
```

  b. Use the **lsbl** command to find the baselines associated with the component:

```
cmd-context  lsbl –component parser@/usr1/tmp/foo_project
07-Sep-99.10:47:47  parser_INITIAL.109  bill   "parser_INITIAL"
  component: parser

07-Sep-99.10:49:06  START.109  bill   "START"
  component: parser
```

  c. Remove the baseline:

```
cmd-context  rmbl –f START.109@/usr1/tmp/foo_project
Removed baseline "START.109@/usr1/tmp/foo_project".
```

  d. Reissue the **rmcomp** command:

```
cmd-context  rmcomp –f parser@/usr1/tmp/foo_project
Removed component "parser@/usr1/tmp/foo_project".
```

SEE ALSO

lscomp, **mkcomp**, **rmbl**

rmdo

Removes a derived object from a VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

- Remove individual derived objects:

  rmdo *do-pname ...*

- Remove collections of derived objects:

  rmdo { **–a·ll** | **–zer·o** } [*pname ...*]

DESCRIPTION

The **rmdo** command deletes one or more *derived objects* (DOs). Use **rmdo** to remove DOs (for example, damaged DOs or DOs that were built incorrectly) so that other users do not use them inadvertently.

NOTE: This command does not apply to *snapshot views*.

The details of the removal process depend on the kind of DO (use **lsdo –long** to determine the kind of DO):

- For a *shared derived object* whose *data container* is in VOB storage, **rmdo** deletes the entry in the VOB database, and also deletes the data container file (from one of the VOB's derived object storage pools).

  CAUTION: If you need to remove a shared DO, use **lsdo –long** to identify the views that reference the DO. Ask the owner of each view to remove the DO from the view with the UNIX **rm(1)** command or by running **make clean** or an equivalent command. If the DO is not removed from the referencing views before you use **rmdo**, error messages appear. For example, when users try to access the DO from the referencing views, the **view_server** logs VOB warnings. Also, you may see `INTERNAL ERROR` messages in the ClearCase **error_log** file; these messages are generated when **clearmake** or an OS-level command tries to access the DO. The derived object's name is removed from the directory by the OS-level access; thus, subsequent accesses return `not found` errors.

- For an *unshared derived object* whose data container is in view-private storage, **rmdo** deletes the entry from the VOB database, but does not delete the data container from view storage. The data container is an ordinary file that can still be listed, executed, and so on, but it cannot be a candidate for configuration lookup. The **ls –long** command lists it with a `[no config record]` annotation. To delete the data file, use the UNIX **rm(1)** command.
- For a *nonshareable derived object*, which does not have an entry in the VOB database, **rmdo** converts the DO into an ordinary view-private file. To delete the file, use the UNIX **rm(1)** command.

In each case, **rmdo** also deletes the associated configuration record if it is no longer needed. Both of the following conditions must be true:

- No other *sibling* DO (created in the same build script execution) still exists.
- The DO is not a build dependency (subtarget) of another DO that still exists.

rmdo does not delete *DO versions*. To delete a DO that has been checked in as a version of an element, use **rmver**.

SCRUBBING OF DERIVED OBJECTS

ClearCase includes a utility, **scrubber**, that deletes shareable DOs. **scrubber** deletes the entries in the VOB database and (for shared DOs) the data containers in the VOB's storage pools. By default, the ClearCase scheduler runs **scrubber** periodically. See the **schedule** reference page for information on describing and changing scheduled jobs.

Each DO pool has scrubbing parameters, which you can modify with the **mkpool –update** command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: DO group member, DO owner, VOB owner, *root* user. To delete a shared DO, you must be the VOB owner or *root* user. See the **permissions** reference page.

Locks: An error occurs if the VOB or pool is locked.

OPTIONS AND ARGUMENTS

HANDLING OF LIKE-NAMED DERIVED OBJECTS. *Default:* Deletes at most one DO for each file name specified with command arguments. A file name with a *DO-ID* (for example, **hello.o@@24–Mar.11:32.412**) specifies exactly which DO to delete. A standard or view-extended pathname specifies the DO that appears in the view.

To determine the DO-IDs of derived objects, use **lsdo**.

–a·ll

Deletes all DOs at a given pathname, regardless of the view they were created in or currently appear in. (However, see the CAUTION on page 852.)

–zer·o
Similar to **–all**, but deletes only those DOs with zero reference counts.

SPECIFYING DERIVED OBJECTS. *Default:* With **–all** or **–zero**, the default is to list all DOs in the current working directory. If you do not specify one of these options, you must supply at least one argument.

do-pname ...
Pathnames of one or more individual DOs. A name with a DO-ID, such as **foo@@10-Nov.10:14.27672**, specifies a particular DO, irrespective of view. A standard UNIX pathname or view-extended pathname specifies the DO that appears in a view.

pname ...
(use with **–all** or **–zero**) One or more standard or view-extended pathnames, each of which can name a file or directory:

- A file name specifies a collection of DOs built at the same pathname.
- A directory name is equivalent to a list of all the file names of DOs built in that directory, including file names that do not currently appear in the view (perhaps after a **make clean**).

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Delete the derived object **hello.o@@24-Mar.11:32.412**.

  cmd-context **rmdo hello.o@@24-Mar.11:32.412**
  ```
  Removed derived object "hello.o@@24-Mar.11:32.412".
  ```

- Delete all derived objects named hello in the current working directory.

  cmd-context **rmdo –all hello**
  ```
  Removed derived object "hello@@23-Mar.14:16.178".
  Removed derived object "hello@@23-Mar.19:25.394".
  ```

- Delete all zero-referenced derived objects in the **hworld** directory.

  cmd-context **rmdo –zero hworld**

```
Removed derived object "hworld/hello.o@@23-Mar.20:42.373".
Removed derived object "hworld/hello.o@@23-Mar.20:36.228".
Removed derived object "hworld/hello@@23-Mar.20:42.382".
Removed derived object "hworld/hello@@23-Mar.20:36.234".
Removed derived object "hworld/util.o@@23-Mar.20:42.376".
Removed derived object "hworld/util.o@@23-Mar.20:36.231".
```

SEE ALSO

clearmake, **lsdo**, **scrubber**

Building Software with ClearCase

rmelem

Removes an element or symbolic link from a VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmelem [**–f·orce**] [**–c·omment** *comment* | **–cfi·le** *comment-file-pname*
| **–cq·uery** | **–cqe·ach** | **–nc·omment**] *pname* ...

DESCRIPTION

The **rmelem** command completely deletes one or more elements or symbolic links. In a snapshot view, **rmelem** also unloads the element from the view.

This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases. In many cases, it is better to use the **rmname** command.

For each element, **rmelem** does the following:

- Removes the entire *version tree* structure from the *VOB database*: element object, branch objects, and version objects.
- Removes all *metadata* items (labels, attributes, hyperlinks, and triggers) that were attached to the element.
- Removes all *event records* for the element.
- (File elements only) Removes the data containers that hold the element's file-system data from its source storage pool.
- Removes all references to the element from versions of the VOB's directory elements. (This means that subsequent listings and comparisons of those directory versions will be historically inaccurate.)
- (Attache only) Removes read-only workspace local files/directories corresponding to successfully removed elements in the view. Local writable files, including any in a directory's subtrees, cause a confirming query to be issued.

- Creates a `destroy element` event record on the element's VOB; this event record is displayed by the **lshistory vob:** command.

RESTRICTION: You cannot remove an element if any of its versions are checked out. (It is not necessary to check out the parent directory before removing one of its elements.)

For each symbolic link, **rmelem** does the following:

- Removes the symbolic link and link object from the VOB.
- Removes all *metadata* items (attributes and hyperlinks) that were attached to the symbolic link .
- Removes all *event records* for the symbolic link.
- Removes all references to the symbolic link from versions of the VOB's directory elements. (This means that subsequent listings and comparisons of those directory versions will be historically inaccurate.)
- (Attache only) Removes read-only workspace local files/directories corresponding to successfully removed symbolic links in the view. Local writable files, including any in a directory's subtrees, cause a confirming query to be issued.

NOTE: **rmelem** does not create an event record when you remove a symbolic link.

rmelem deletes an *instance* of an element type object. To delete the element type object itself or to delete the type object and all its instances, use the **rmtype** command.

Deleting a Directory Element

NOTE: Only dynamic views have **lost+found** directories and derived objects.

Deleting a directory element may cause some other elements (and symbolic links, if the VOB is replicated) to be *orphaned*: no longer cataloged in any version of any directory. **rmelem** displays a message and moves an orphaned element or symbolic link to the VOB's **lost+found** directory:

```
cleartool: Warning: Object "foo.c" no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory
as "foo.c.a0650992e2b911ccb4bc08006906af65".
```

(See the **mkvob** reference page for a description of this directory.)

Each derived object in the deleted directory is also moved to **lost+found**. The derived object has no data, but you can use it in such commands as **lsdo** and **catcr**. View-private objects in the deleted directory are temporarily stranded, but can be transferred to the view's own **lost+found** directory, as follows:

1. Use **lsprivate** to locate stranded files and to determine the ClearCase identifier of the deleted directory element:

cmd-context **lsprivate –invob /tmp/david_phobos_hw**

.
.
.

```
#<Unavailable-VOB-1>/<DIR-c8051152.e2ba11cc.b4c0.08:00:69:06:af:65>/myfile
```

2. Use **recoverview** to move all the stranded files for the deleted directory:

   cmd-context **recoverview –dir c8051152.e2ba11cc.b4c0.08:00:69:06:af:65 –tag myview**

   ```
   Moved file /usr/people/david/myview.vws/.s/lost+found/5ECC880E.00A5.myfile
   ```

Deleting Elements and Symbolic Links from the lost+found Directory

Use **rmelem** to delete unwanted elements or symbolic links from the **lost+found** directory (**lost+found** is associated with dynamic views only). If you need an element in **lost+found**, catalog it in a versioned directory using **mv**.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element owner, symbolic link owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, pool (nondirectory elements only).

OPTIONS AND ARGUMENTS

CONFIRMATION STEP. *Default:* **rmelem** prompts for confirmation before deleting anything.

–f·orce
(ClearCase and ClearCase LT only) Suppresses the confirmation step.

(Attache only) Suppresses the confirmation step for deleting anything in the view or VOB. The confirmation for local writable files still pertains.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE ELEMENTS TO BE REMOVED. *Default:* None.

pname ...
One or more pathnames, indicating the elements or symbolic links to be deleted. An extended pathname to a particular version or branch of an element references the element itself.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Delete the file element **rotate.c**.

cmd-context **rmelem rotate.c**
```
Element "rotate.c" has 1 branches, 2 versions, and is entered
in 6 directory versions.
Remove element, all its branches and versions and modify all directory
versions containing element? [no] yes
Removed element "rotate.c".
```

- Delete the directory element **release**. Note that an orphaned element, **hello**, is moved to the VOB's **lost+found** directory. (The view context is dynamic.)

  cmd-context **rmelem release**
  ```
  Element "release" has 1 branches, 9 versions, and is entered
  in 35 directory versions.
  Remove element, all its branches and versions and modify all directory
  versions containing element? [no] yes
  cleartool: Warning: Object "hello" no longer referenced.
  Object moved to vob lost+found directory as
   "hello.5d400002090711cba06a080069061935".
  Removed element "release".
  ```

- Delete the symbolic link **text.c** from the **lost+found** directory. (The view context is dynamic.)

```
cmd-context rmelem /vobs/dev/lost+found/text.c
CAUTION! This will destroy the symbolic link, and will remove the
symbolic link from all directory versions that now contain it.  Once you
destroy the  symbolic link, it will be hard to restore it to its current
state. If you want to preserve the symbolic link, but remove references
to it from future directory versions, use the “rmname” command.
Symbolic link ”text.c” is entered in 3 directory versions.
Destroy symbolic link? yes
Removed symbolic link ”text.c”.
```

SEE ALSO

mkelem, **mkvob**, **rmbranch**, **rmname**, **rmtype**, **rmver**

rmfolder

Remove a UCM folder

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

rmfolder [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–f·orce**] *folder-selector ...*

DESCRIPTION

The **rmfolder** command deletes one or more UCM folders. You cannot delete a folder if it contains any projects, other folders, or is the **RootFolder**.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the folder's owner, the VOB owner, or **root**.

Locks: An error occurs if there are locks on any of the following objects: the folder.

Mastership: The current replica must master the folder.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the specifed folder is to be deleted.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE FOLDER. *Default*: None.

folder-selector ...
Specifies one or more folders to delete.

folder-selector is of the form: [**folder:**]*folder-name*[*@vob-selector*] and *vob* is the folder's UCM project VOB.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove a folder that contains a subfolder, moving the subfolder to a new location.

  a. Issue the **rmfolder** command:

```
cmd-context rmfolder –f top
cleartool: Error: Cannot remove folder that has sub projects or folders.
cleartool: Error: Unable to remove folder "top".
```

  b. Use **lsfolder** to find subprojects or folders for the specified folder:

```
cmd-context lsfolder –l top
folder "top"
 07-Sep-99.10:20:08 by Smith
  "My Top Level Folder."
  owner: Smith
  group: user
  title: Top
  contains folders:
    parsers
  contains projects:
```

  c. Move the subfolder to a new location:

```
cmd-context chfolder –to RootFolder parsers
Changed folder "parsers".
```

  d. Reissue the **rmfolder** command:

```
cmd-context rmfolder top
Remove folder "top"?  [no] yes
Removed folder "top".
```

SEE ALSO

chfolder, **lsfolder**, **mkfolder**, **rmproject**

rmhlink

Removed a hyperlink object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmhlink [–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment]** *hlink-selector ...*

DESCRIPTION

The **rmhlink** command removes one or more *hyperlinks* from VOB-database objects. Hyperlinks can be attached to objects by the **mkhlink** command and by triggers (**mktrtype –mkhlink**). See the **mkhlink** reference page for a list of objects to which hyperlinks can be attached.

rmhlink deletes a reference to a hyperlink type object. To delete the hyperlink type object itself or the type object and all its instances, use the **rmtype** command.

To list existing hyperlinks, use the **describe** command, or use the **find** command with the **hltype** primitive.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, object group member, object owner, VOB owner, or **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type. For non-file-system objects, an error occurs if the VOB, object, object type, or hyperlink type is locked.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE HYPERLINKS TO BE REMOVED. *Default:* None.

hlink-selector ...

One or more names of hyperlink objects, in this form:

*hyperlink-type-name***@***hyperlink-ID*[**@***pname-in-vob*]

Hyperlinks are not file system objects; you cannot specify them with command interpreter wildcards. The final component is required only for a hyperlink in another VOB. For example:

```
DesignFor@598f
RelatesTo@58843@/vobs/monet
```

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove a hyperlink of type **tested_by** from the element **cm_add.c**. Use **describe** to determine the hyperlink selector.

  cmd-context **describe –long cm_add.c@@**

  ```
  file element "cm_add.c@@"
   created 08-Dec-98.12:12:52 by Chuck Jackson (test user)
  (jackson.dvt@oxygen)
   element type: c_source
   Protection:
    User : jackson  : r-x
    Group: dvt      : r-x
    Other:          : r-x
   source pool: sdft cleartext pool: cltxt2
   Hyperlinks:
    tested_by@714@/usr/hw /usr/hw/src/cm_add.c@@
     "edge effects" -> /usr/hw/src/edge.sh@@ "regression A"
  ```

  cmd-context **rmhlink tested_by@714**

  ```
  Removed hyperlink "tested_by@714".
  ```

- Remove two hyperlinks from the **src** directory. Use **describe** to determine the hyperlink selectors.

  cmd-context **describe –long src**

```
directory version "src@@/main/9"
 created 08-Dec-98.12:23:46 by Chuck Jackson (test user)
(jackson.dvt@oxygen)
 Element Protection:
  User : jackson  : r-x
  Group: dvt      : r-x
  Other:          : r-x
 element type: directory
 Hyperlinks:
 h3@1320@/usr/hw /usr/hw/src@@/main/9 ->
 h1@1324@/usr/hw /usr/hw/src/hello@@/main/1 -> /usr/hw/src@@/main/9
 h2@1329@/usr/hw /usr/hw/bin@@/main/1 -> /usr/hw/src@@/main/9
```

cmd-context **rmhlink h1@1324 h2@1329**

```
Removed hyperlink "h1@1324".
Removed hyperlink "h2@1329".
```

SEE ALSO

describe, **lshistory**, **mkhlink**, **rmtype**

rmlabel

Removes a version label from a version

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmlabel [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery**
| **–cqe·ach** | **–nc·omment**]
[**–ver·sion** *version-selector*] *label-type-selector pname ...*

DESCRIPTION

The **rmlabel** command removes one or more *version labels* from versions of elements. Labels can be attached to versions by the **mklabel** command and by *triggers* (**mktrtype –mklabel**).

rmlabel deletes a reference to a label type object. To delete the label type object itself or the type object and all its instances, use the **rmtype** command.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, label type.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE VERSIONS TO BE UNLABELED. *Default:* None.

pname ...
One or more pathnames, indicating versions from which the label is to be removed. What kind of pathname is valid depends on how the label has been used:

If the label has been used only once in an element's version tree, you can specify the element itself, or any of its branches or versions:

foo.c	*(version selected by view)*
foo.c@@	*(element itself)*
foo.c@@/main/rel2_bugfix	*(branch of element)*

If the label has been used multiple times, you must specify either the version to which the label is attached, or the branch on which that version resides.

foo.c	*(version selected by view)*
foo.c@@/REL1	*(version specified by label)*
foo.c@@/main/rel2_bugfix/3	*(version specified by version-ID)*
foo.c@@/main/rel2_bugfix	*(branch on which version resides)*

Using the **–version** option modifies the way in which this argument is interpreted.

–ver·sion *version-selector*
Specifies the version from which the label is to be removed. See the **version_selector** reference page for syntax details. Using this option overrides a version-extended pathname. For example:

cmd-context **rmlabel XXX util.c@@/REL1**	*(removes label from version REL1)*
cmd-context **rmlabel -ver /main/3 XXX util.c@@/REL1**	*(removes label from version /main/3)*

SPECIFYING THE LABEL TO BE REMOVED. *Default:* None.

label-type-selector
An existing label type. Specify *label-type-selector* in the form [**lbtype:**]*type-name*[**@***vob-selector*]

type-name	Name of the label type
vob-selector	VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove the label **REL3** from a version of **msg.c** without specifying which version (assumes the label is attached to one version only).

  cmd-context **rmlabel REL3 msg.c**

  ```
  Removed label "REL3" from "msg.c" version "/main/1".
  ```

- Remove the label **REL2** from the version of element **util.c** specified by a version selector.

  cmd-context **rmlabel –version /main/REL2 REL2 util.c**

  ```
  Removed label "REL2" from "util.c" version "/main/1".
  ```

- Remove the label **REL1.1** from version 1 on the **maintenance** branch of file element **util.c**. Use a version-extended pathname to indicate the version.

  cmd-context **rmlabel REL1.1 util.c@@/main/maintenance/1**

  ```
  Removed label "REL1.1" from "util.c" version "/main/maintenance/1".
  ```

SEE ALSO

lstype, **mklabel**, **rename**, **rmtype**

rmmerge

Removes a merge arrow from an element's version tree

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmmerge [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery**
| **–cqe·ach** | **–nc·omment**]
from-pname to-pname

DESCRIPTION

The **rmmerge** command deletes an existing merge arrow (a hyperlink of the predefined type **Merge**) between two versions of an element. Thus, this command is a specialized form of the **rmhlink** command. The two commands have an identical result; they differ only in the way you specify the merge arrow:

- With **rmhlink**, you specify the merge arrow itself, using a hyperlink selector.
- With **rmmerge**, you specify the versions linked by the merge arrow.

To list existing merge arrows, use the **describe** command, or use the **find** command with the **hltype** primitive. For example:

cmd-context **describe util.c**

```
version "util.c@@/main/3"
 created 05-Apr-99.17:01:12 by Allison (akp.user@starfield)
 element type: text_file
 Hyperlinks:
 Merge@148@/usr/tmp/poolwk
 /usr/tmp/poolwk/src/util.c@@/main/rel2_bugfix/1 ->
 /usr/tmp/poolwk/src/util.c
```

Renaming the Merge Hyperlink Type

Renaming the predefined hyperlink type for merge arrows does not defeat **rmmerge**. You specify the element's versions; **rmmerge** then determines the hyperlink type used for merge arrows in that element's VOB.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, hyperlink type.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE VERSIONS. *Default:* None.

from-pname, *to-pname*
Extended pathnames of the versions connected by the merge arrow. The order in which you specify the versions is important: the source version first, the target version second.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove the merge arrow between the latest version on the **rel2_bugfix** branch and the version of **util.c** in the view.

  cmd-context **rmmerge util.c@@/main/rel2_bugfix/LATEST util.c**
  ```
  Removed merge from "util.c@@/main/rel2_bugfix/1" to "util.c".
  ```

SEE ALSO

merge, **rmhlink**

rmname

Removes the name of an element or VOB symbolic link from a directory version

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rm·name [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery**
| **–cqe·ach** | **–nc·omment**]
[**–nco** [**–f·orce**]] *pname ...*

DESCRIPTION

By default, a name can be removed from a directory only if that directory is checked out. **rmname** appends an appropriate line to the directory's checkout comment.

rmname is similar to the UNIX **unlink(2)** system call: it modifies one or more checked-out directories by removing the names of *elements* and/or *VOB symbolic links*. Old versions of the directories do not change; the names continue to be cataloged in the old versions.

To remove a name from a checked-in directory version, you can use the **–nco** option. For example, you may want to remove an old symbolic link that points to a file that has been removed. You must be the VOB owner or **root** to use this option, and the VOB must not be replicated.

In Attache, for all successfully removed names in the view, any corresponding read-only local files and directories are deleted in the workspace; local writable files, including any in a directory's subtrees, cause a confirming query to be issued.

In a *snapshot view*, this command implicitly executes an **update** operation on the affected elements.

Example: Suppose you checked out version 3 of a directory named **a.dir**. Only your view or workspace sees this directory version while it is checked out. The command **rmname foo.c** deletes the name **foo.c** from the checked-out version of the directory and from your Attache workspace, but leaves references to **foo.c** in earlier versions (if any) intact. When you check in the directory, all views can access the new version 4, which does not include **foo.c**.

Keep the following points in mind:

- **rmname** does not delete elements themselves, only references to elements. Use **rmelem** (very carefully) to delete elements and all their names from their VOBs.
- Removing the last reference to an element name causes the element to be *orphaned*. Such elements are moved to the VOB's **lost+found** directory. (See the **mkvob** command for details.)
- Removing the last reference to a VOB symbolic link works differently depending on whether the VOB is replicated:
  - If the VOB is unreplicated, the link object is deleted.
  - If the VOB is replicated, the link object is moved to the VOB's **lost+found** directory.

Undoing the rmname Command

To restore a directory entry for an element that has been removed with **rmname**, use the **ln** command to create a *VOB hard link* to the element's entry in any previous version of the directory. For example:

cmd-context **checkout src**	*(checkout parent directory)*
cmd-context **rmname src/msg.c**	*(oops!)*
cmd-context **ln src@@/main/LATEST/msg.c src/msg.c**	*(restore deleted name)*

If there are no entries for the element in any previous version of the directory, the element is *orphaned*; ClearCase or Attache has moved it to its VOB's **lost+found** directory. You can move/rename the element to its proper location with the **cleartool** or Attache **mv** command. (You cannot use **ln** to link elements that are in the **lost+found** directory.)

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required if directory is checked out; see **checkout** permissions. To use the **–nco** option, you must be one of the following: VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* **–nc**. Creates one or more *event records*, with commenting controlled by your home directory's **.clearcase_profile** file (ClearCase and ClearCase LT) or your remote home directory's **.clearcase_profile** file (Attache). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

REMOVING A NAME FROM A CHECKED-IN DIRECTORY VERSION. *Default:* You must check out a directory to remove a name and/or VOB symbolic link from it.

–nco [**–f·orce**]
Prompts for confirmation, then removes the name or link from the checked-in directory version that you specify. Use the **–force** option to suppress the confirmation step.

NOTE: You cannot use **–nco** in a replicated VOB.

SPECIFYING THE NAMES TO BE REMOVED. *Default:* None.

pname ...
One or more pathnames, specifying the elements and/or VOB symbolic links whose names are to be removed from their parent directory. In ClearCase and ClearCase LT, you can specify an element itself, or any of its branches or versions.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: Examples assume that the current working directory is checked out.

- Delete the name **util.c** from the current directory version. (In Attache, this also removes the local writable file **util.c** from the workspace.)

  cmd-context **rmname util.c**

  ```
  Removed "util.c".
  ```

- Delete the last reference to the directory element **subd** from the current directory version.

  cmd-context **rmname subd**

  ```
  cleartool: Warning: Object "subd" no longer referenced.
  Object moved to vob lost+found as
   "subd.5a200007ed11f0d709066505efe922a8".
  Removed "subd".
  ```

- As *root* user, delete the name **hello.h** from the directory version **.@@/main/2**.

  cmd-context **rmname –nco –force .@@/main/2/hello.h**

  ```
  Removed ".@@/main/2/hello.h".
  ```

SEE ALSO

ln, **mv**, **rmelem**, **rmver**, **unlink(2)**, **update**

rmpool

Removes a VOB storage pool

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

rmpool [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
pool-selector ...

DESCRIPTION

The **rmpool** command deletes one or more *storage pool* directories from a VOB, along with all the *data container* files stored within them.

Restrictions

Before removing a storage pool, you must reassign all its currently assigned elements to a different pool, using the **chpool** command. Otherwise, **rmpool** aborts with an `elements using pool` error. To list all the elements in a source or cleartext pool, use a **find** command. For example:

cmd-context **find -all -element 'pool(source_2)' –print**

This command does not work with derived object pools.

Deleting Derived Object Pools

There is no way to move a shared derived object from one pool to another. Thus, you can delete a derived object pool only if either condition is true:

- No directory elements have been assigned to the pool.
- All data containers in the pool have been removed by the **scrubber** program or **rmdo** commands, and each directory element that currently uses the pool has been assigned to a different derived object pool.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: pool owner, VOB owner, *root* user. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, pool.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE POOLS TO BE REMOVED. *Default:* Removes a pool from the VOB containing the current working directory unless you specify another VOB with the **@***vob-selector* suffix.

pool-selector ...
One or more names of existing storage pools. Specify *pool-selector* in the form [**pool:**]*pool-name*[**@***vob-selector*]

pool-name	Name of the storage pool See the *Object Names* section in the **cleartool** reference page for rules about composing names.
vob-selector	VOB specifier Specify *vob-selector* in the form [**vob:**]*pname-in-vob* *pname-in-vob* — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Change all elements using the **c_source_pool** to use the default source pool (**sdft**) instead. Then, delete **c_source_pool**.

  cmd-context **find . –all –element 'pool(c_source_pool)' **
  –exec 'cleartool chpool –force sdft $CLEARCASE_PN'

```
Changed pool for "/usr/hw/src" to "sdft".
Changed pool for "/usr/hw/src/libutil.a" to "sdft".

.
.
.
```

cmd-context **rmpool c_source_pool**

```
Removed pool "c_source_pool".
```

SEE ALSO

describe, **chpool**, **find**, **lspool**, **mkpool**, **rmdo**, **rename**, **scrubber**

rmproject

Removes a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

rmproj·ect [–c·omm ent *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–nc·omment**]
[**–f·orce**] *project-selector ...*

DESCRIPTION

The **rmproject** command deletes one or more UCM projects.

All streams must be removed before deleting a project. You cannot delete a project that contains a stream.

ClearQuest-Enabled Projects

When you delete a project that uses the UCM-ClearQuest integration, the project is unlinked from its associated ClearQuest record, but the ClearQuest record is not deleted.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the project's owner, the VOB owner, or **root**.

Locks: An error occurs if there are locks on any of the following objects: the project.

Mastership: The current replica must master the project.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the specifed project is to be deleted.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE PROJECT. *Default:* None.

project-selector ...
Specifies one or more projects to delete.

project-selector is of the form: [**project:**]*project-name*[*@vob-selector*] and *vob* is the project's UCM project VOB.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove a project that contains a stream.

  a. Issue the **rmproject** command:

```
cmd-context rmproject html_parser
Remove project "html_parser"?  [no] yes
cleartool: Error: Cannot remove project that has streams.
cleartool: Error: Unable to remove project "html_parser".
```

  b. Use **lsproject –long** to see a detailed description of the project, including a list of any streams contained by the project:

```
cmd-context lsproject -long html_parser
cleartool lsproject -l html_parser
project "html_parser"
 07-Sep-99.11:24:27 by Bsmith
  owner: bsmith
  group: user
  folder: parsers
  title: html_parser
  integration stream: html_parser_int
  development streams:
    html_parser_int
  modifiable components:
  default rebase promotion level: INITIAL
  recommended baselines:
```

  c. Remove the stream. The **–force** option bypasses the confirmation step.

```
cmd-context rmstream -force html_parser_int
Removed stream "html_parser_int".
```

  d. Reissue the **rmproject** command:

cmd-context **rmproject -force html_parser**
```
Removed project "html_parser".
```

SEE ALSO

lsproject, **lsstream**, **mkproject**, **rmstream**

rmregion

Unregisters a ClearCase network region

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

rmregion –tag *region-tag* [**–rma·ll** [**–pas·sword** *tag-registry-password*]]

DESCRIPTION

The **rmregion** command removes a region entry from the ClearCase registry's **regions** file.

rmregion modifies the ClearCase registry only. It does not affect client host region assignments. If you remove a region to which ClearCase client hosts are assigned, those clients receive error messages.

To reassign a client host to a new region, run a command like the following on the client host:

echo "region2" > /var/adm/atria/rgy/rgy_region.conf

See **registry_ccase** and *Administering ClearCase* for more information on ClearCase network regions.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE REGION TAGS. *Default:* None. You must specify the name of the region to unregister.

–tag *region-tag*
Specifies a region to unregister.

–rma·ll [**–pas·sword** *tag-registry-password*]
Removes the region specified with **–tag**, along with any view-tags and VOB-tags in that region. If the region contains VOB-tags, you must supply the VOB-tag registry password (either with the **–password** option or at the prompt).

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove region **devel3** from the ClearCase registry.

  cmd-context **rmregion –tag devel3**

- Remove all tags for the **test1** region.

  cmd-context **rmregion –tag test1 –rmall**

FILES

/var/adm/atria/rgy/regions
/var/adm/atria/rgy/rgy_region.conf

SEE ALSO

mkregion, **registry_ccase**

rmstgloc

Removes registry entries for server storage locations.

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

- ClearCase only:

  rmstgloc [**–all**] [**–reg·ion** *network-region*] { *stgloc-name* | **–sto·rage** *stgloc-pname* }

- ClearCase LT only:

  rmstgloc { *stgloc-name* | **–sto·rage** *stgloc-pname* }

DESCRIPTION

The **rmstgloc** command deletes registrations for view and VOB server storage location registrations for views and VOBs. The associated physical storage is not deleted, and views and VOBs residing at the server storage location continue to be accessible. However, no views or VOBs may be created at the server storage location after you have removed its registry entries.

To remove view or VOB physical storage (and their registrations), always use **rmview** or **rmvob**, never an operating system command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. *Other restrictions:* The specified server storage locations must not contain any views or VOBs.

OPTIONS AND ARGUMENTS

SPECIFYING ALL QUALIFYING SERVER STORAGE LOCATIONS. *Default:* None.

–all

Deletes all server storage locations that are selected by other options and arguments you specify. For example, **rmstgloc –all** *stgloc-name* deletes all server storage locations with names that match *stgloc-name*, regardless of region.

SPECIFYING THE NETWORK REGION. *Default:* The local host's network region. (Use the **hostinfo –long** command to display the network region.) See the **registry_ccase** reference page for a discussion of *network regions*.

–reg·ion *network-region*
Specifies a network region where a server storage location that is to be deleted resides. An error occurs if the region does not already exist.

SPECIFYING THE SERVER STORAGE LOCATION. *Default:* None.

stgloc-name
Unregisters the server storage location with the specified name.

–sto·rage *stgloc-pname*
Unregisters the server storage location specified by the given path.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove the server storage location named **stgloc_vob1**.

  cmd-context **rmstgloc stgloc_vob1**
  ```
  cleartool: Warning: The storage location has only been removed from the
  ClearCase registry. You must manually remove the physical storage location
  directory.
  ```

SEE ALSO

lsstgloc, **mkstgloc**, **mkview**, **mkvob**, **registry_ccase**

rmstream

Remove a UCM stream

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

rmstream [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**] [**–f**·**orce**] *stream-selector* ...

DESCRIPTION

The **rmstream** command deletes one or more UCM streams.

The following restrictions apply:

- The stream cannot contain activities.
- The stream can have no baselines other than the set of initial baselines associated with it.
- No views can be attached to the stream.

In addition, a project's integration stream cannot be removed while other project streams exist.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the stream owner, the VOB owner, or **root**.

Locks: An error occurs if there are locks on any of the following objects: the stream.

Mastership: The current replica must master the stream.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**
Overrides the default with the option you specify. See the **comments** reference page.

CONFIRMATION STEP. *Default:* Prompts for confirmation that the specifed stream is to be deleted.

–f·**orce**
Suppresses the confirmation step.

SPECIFY THE STREAM TO BE REMOVED. *Default:* None.

stream-selector ..
Specifies one or more streams to delete.

You can specify the stream as a simple name or as an object selector of the form **[stream]:***name***@***vob-selector*, where *vob-selector* specifies a project VOB (see the **cleartool** reference page). If you specify a simple name and the current directory is not a project VOB, then this command assumes the stream resides in the project VOB associated with the current view. If the current directory is a project VOB, then that project VOB is the context for identifying the stream.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove a stream that has a view attached to it.
  a. Issue the **rmstream** command. You are told that the stream cannot be removed because a view is still attached to it:

```
cmd-context rmstream -f html_parser_int
cleartool: Error: Cannot remove stream that has a view
("html_parser_int_view") attached to it.

cleartool: Error: Unable to remove stream "html_parser_int".
```

  b. Display a description of the stream to see what views are associated with it:

```
cmd-context describe stream:html_parser_int stream "html_parser_int"
  created 11-Sep-99.11:27:01 by JFMuggs
  owner: jfm
  group: user
  project: html_parser
  title: html_parser_int
  contains activities:
  foundation baselines:
  views:
    html_parser_int_view

  Guarding: brtype:html_parser_int@/usr1/tmp/foo_project
```

  c. Remove the view:

```
cmd-context rmview -tag html_parser_int_view
Removing references from VOB "/usr1/tmp/foo_project" ...
```

```
Removed references to view "/net/propane/usr1/tmp/html_parser_int.vws"
from VOB "/usr1/tmp/foo_project".
```

d. Reissue the **rmstream** command:

cmd-context **rmstream -f html_parser_int**
```
Removed stream "html_parser_int".
```

SEE ALSO

lsstream, **mkstream**

rmtag

Removes a view-tag or a VOB-tag from the networkwide storage registry

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only—Remove a view-tag:

  rmtag –vie·w [**–reg·ion** *network-region* | **–a·ll**] *view-tag* ...

- ClearCase and Attache only—Remove a VOB-tag:

  rmtag –vob [**–reg·ion** *network-region* | **–a·ll**]
  [**–pas·sword** *tag-registry-password*] *vob-tag* ...

- ClearCase LT only—Remove a view- or VOB-tag:

  rmtag { **–vie·w** *view-tag* ... | **–vob** *vob-tag* ... }

DESCRIPTION

The **rmtag** command removes one or more entries from the network's view-tag registry or vob-tag registry. See the **registry_ccase** reference page for a discussion of the registries. You cannot remove a tag that is currently in use.

CleasrCase and Attache Only—Using rmtag

A VOB-tag is in use if the VOB is active on any host in the network region. Use the **cleartool** or Attache **umount** command to deactivate a VOB on all hosts in the region before removing its tag.

A view-tag for a dynamic view is in use if any user process is set to the view specified by this tag, or if any user process has a current working directory that is a *view-extended pathname* based on this tag.

A VOB or view must always have a tag in its home region: the network region of the host where the VOB or view storage directory physically resides. If you remove a home-region tag, create a new one immediately.

You must supply the network's *VOB-tag password* when deleting a public VOB-tag; if you don't use the **–password** option, you are prompted for the password. See the **rgy_passwd** and **registry_ccase** reference pages for information on the VOB-tag password.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE KIND OF TAG. *Default:* None.

–vie·w
Removes one or more view-tags.

–vob
Removes one or more VOB-tags.

SPECIFYING A NETWORK REGION. *Default:* Removes tags that are defined for the local host's network region. (Use the **hostinfo –long** command to list a host's network region.) See the **registry_ccase** reference page for a discussion of *network regions*.

–reg·ion *network-region*
Removes a tag defined for the specified *network region*. An error occurs if the region does not already exist.

–a·ll
Removes a tag from all network regions for which it is defined.

SPECIFYING THE VOB-TAG PASSWORD. *Default:* If you attempt to remove a public VOB-tag, **rmtag** prompts you for the VOB-tag password. (See also **rgy_passwd**.)

–pas·sword *tag-registry-password*
Specifies the password on the command line.

CAUTION: This is a potential security breach, because the password remains visible on your display buffer.

SPECIFYING THE TAGS. *Default:* None.

view-tag ..
One or more view-tags to be removed.

vob-tag ..
One or more VOB-tags to be removed.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Remove the view-tag **R2alpha** from the view registry.

cmd-context **rmtag –view R2alpha**

- Remove the VOB-tag **/vobs/tests** from all network regions.

  cmd-context **rmtag –vob –all /vobs/tests**

SEE ALSO

mktag, **mkview**, **mkvob**, **registry_ccase**, **rgy_passwd**, **rmview**, **rmvob**

rmtrigger

Removes trigger from an element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmtrigger [–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery**
| **–cqe·ach** | **–nc·omment]**
[–nin·herit | **–nat·tach] [–r·ecurse]** *trigger-type-selector pname ...*

DESCRIPTION

The **rmtrigger** command removes an attached *trigger* from one or more elements. By default, **rmtrigger** removes the trigger from both the *attached list* and the *inheritance list* (if a directory element). You can modify the default action for directory elements with the **–ninherit** and **–nattach** options.

The specified *trigger-type-selector* is not affected by **rmtrigger**. To delete the trigger type, use the **rmtype** command. Note that you can remove an attached trigger from an element even if the trigger type is *obsolete*.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, or **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, trigger type.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

MANIPULATING THE TRIGGER LISTS OF A DIRECTORY ELEMENT. *Default:* The trigger is removed from both of a directory element's trigger lists: its *attached list* and its *inheritance list*.

–nin·herit
(Directory element only) The trigger is removed from the directory's attached list, but remains on its inheritance list. The trigger does not fire when the monitored operation is performed on the directory itself, but new elements created in that directory inherit the trigger.

–nat·tach
(Directory element only) The trigger is removed from the directory's inheritance list, but remains on its attached list. The trigger continues to fire when the monitored operation is performed on the directory itself, but new elements created in that directory do not inherit the trigger.

REMOVING TRIGGERS FROM AN ENTIRE SUBDIRECTORY TREE. *Default:* If a *pname* argument names a directory element, the trigger is removed only from the element itself, not from any of the existing elements within it.

–r·ecurse
Processes the entire subtree of each *pname* that is a directory element (including *pname* itself). VOB symbolic links are *not* traversed during the recursive descent into the subtree.

SPECIFYING THE TRIGGER TYPE. *Default:* None.

trigger-type-selector
The name of an existing *element* trigger type. Specify *trigger-type-selector* in the form [**trtype:**]*type-name*[**@***vob-selector*]

type-name — Name of the trigger type
vob-selector — VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

SPECIFYING THE ELEMENTS. *Default:* None.

pname ...
One or more pathnames, specifying elements from which triggers (instances of the specified trigger type) are to be removed.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Remove an attached trigger from **hello.c**.

  cmd-context **rmtrigger trig1 hello.c**
  ```
  Removed trigger "trig1" from attached list of "hello.c".
  ```

- Remove an attached trigger from the **src** directory's attached list, but leave it in the inheritance list.

  cmd-context **rmtrigger –ninherit trig1 src**
  ```
  Removed trigger "trig1" from attached list of "src".
  ```

- Remove an attached trigger from the **release** directory's inheritance list, but leave it in the attached list.

  cmd-context **rmtrigger –nattach trig1 release**
  ```
  Removed trigger "trig1" from inheritance list of "release".
  ```

SEE ALSO

describe, **mktrigger**, **mktrtype**, **rmtype**, **unlock**

rmtype

Removes a type object from a VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmtype [**–ign·ore**] [**–rma·ll** [**–f·orce**]]
[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
type-selector ...

DESCRIPTION

The **rmtype** command removes one or more *type objects* from a VOB.

RESTRICTION: You cannot remove a type object if there are any instances of that type. For example, if any version of any element is labeled **REL1**, you cannot remove the **REL1** label type (unless you specify the **–rmall** option).

RESTRICTION: You cannot remove an element type from a replicated VOB.

The file **vista.tjf** records updates to the VOB that result from **rmtype** operations. **vista.tjf** can grow very large. To limit its size, read about the file **db.conf** in the **config_ccase** reference page.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: type owner, VOB owner, or **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, type.

OPTIONS AND ARGUMENTS

REMOVING INSTANCES OF THE TYPE. *Default:* If there are any instances of a specified type object, **rmtype** does not remove the type object.

–rma·ll
Removes all instances of a type, and then proceeds to remove the type object itself. If the type object is a global type, or is a local copy of a global type, **rmtype** removes the global type and all local copies of the type.

CAUTION: If the **rmtype –rmall** command fails for any reason, you must address the causes of the failure and enter the command again. You must persist until the command completes successfully and the type is removed. Failure to do so will result in inconsistent metadata.

CAUTION: This option potentially destroys a great deal of data.

–f·orce (for use with **–rmall** only)
By default, **rmtype** prompts for confirmation when you use the **–rmall** option to request removal of all instances of a type. The **–force** option suppresses the confirmation step.

–ign·ore (for use with trigger types only)
Removes a trigger type even if a previously defined preoperation trigger would otherwise prevent it from being removed.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE TYPE OBJECTS TO BE REMOVED. *Default:* Removes types from the VOB that contains the current working directory unless you specify another VOB with the *@vob-selector* suffix.

type-selector ...
One or more names of existing type objects, of the specified kind. Specify *type-selector* in the form *type-kind***:***type-name*[**@***vob-selector*]

type-kind oOne of

attype	Attribute type
brtype	Branch type
eltype	Element type
hltype	Hyperlink type
lbtype	Label type
trtype	Trigger type

type-name Name of the type object

vob-selector VOB specifier

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Delete the branch type **patch3**.

cmd-context **rmtype brtype:patch3**
```
Removed attribute type "patch3".
```

- Delete the attribute type **QA_date** in the VOB **/vobs/tests**.

  cmd-context **rmtype attype:QA_date@/vobs/tests**
  ```
  Removed attribute type "QA_date".
  ```

- Delete all branches of type **expmnt3** (along with all the versions on those branches and any subbranches); then delete the **expmnt3** branch type itself.

  cmd-context **rmtype –rmall brtype:expmnt3**
  ```
  There are 1 branches of type "expmnt3".
  Remove branches (including all sub-branches and sub-versions)? [no] yes
  Removed branches of type "expmnt3".
  Removed branch type "expmnt3".
  ```

- Delete the hyperlink type **design_doc**.

  cmd-context **rmtype hltype:design_doc**
  ```
  Removed hyperlink type "design_doc".
  ```

- Remove all instances of the label type **REL2**; then delete the label type.

  cmd-context **rmtype –rmall lbtype:REL2**
  ```
  There are 7 labels of type "REL2".
  Remove labels? [no] yes
  Removed labels of type "REL2".
  Removed label type "REL2".
  ```

- Delete the trigger type **trig1**. Use the **–ignore** option to ensure that the command executes without interference from a previously defined trigger.

cmd-context **rmtype –ignore trtype:trig1**
```
Removed trigger type "trig1".
```

SEE ALSO

config_ccase, **describe**, **lshistory**, **lstype**, **mkattype**, **mkbrtype**, **mkeltype**, **mkhltype**, **mklbtype**, **mktrtype**, **rename**

rmver

Removes a version from the version tree of an element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmver [**–f·orce**] [**–xbr·anch**] [**–xla·bel**] [**–xat·tr**] [**–xhl·ink**] [**–dat·a**]
[**–ver·sion** *version-selector* | **–vra·nge** *low-version high-version*]
[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
pname ...

DESCRIPTION

This command destroys information irretrievably. Using it carelessly may compromise your organization's ability to support old releases.

rmver deletes one or more versions from their elements. For each version, this entails the following:

- Removal of the *version object* from the VOB database
- Removal of all metadata items (labels, attributes, hyperlinks, and triggers) that were attached to the deleted version
- Removal of all *event records* for the deleted version
- (File elements only) Removal of the data containers that hold the deleted version's file system data

A `destroy version` event record is created for the element.

In general, a removed version is physically deleted from the VOB source pool. However, a removed version is logically deleted if it has a descendant and is managed by the **z_text_file_delta** or **text_file_delta** type managers. See the **type_manager** reference page for more information on the type managers.

Behavior in Snapshot Views

In a *snapshot view*, **rmver** does not unload the element, but leaves a view-private copy of the element in the view. In other respects, **rmver** behaves the same in a snapshot view as it does in a *dynamic view*.

Restrictions

You cannot delete a version from which someone currently has a checkout. You cannot delete version 0 on a branch, except by deleting the entire branch. (See **rmbranch**.)

Deleted Version-IDs

The version-ID of a deleted version is never reused. There is no way to collapse a branch to fill the gaps left by deleted versions. If a deleted version was the last version on a branch (say, version 6), the next checkin on that branch creates version 7.

A reference to a deleted version produces a `not found` or `no such file or directory` error.

Controlling the Size of the vista.tjf File

The file **vista.tjf** records updates to the VOB that result from **rmver** operations. **vista.tjf** can grow very large. To limit its size, read about the file **db.conf** in the **config_ccase** reference page.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: version creator, element owner, VOB owner, or **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch, pool (non-directory elements only).

OPTIONS AND ARGUMENTS

CONFIRMATION STEP. *Default:* **rmver** prompts for confirmation before deleting anything.

–f·orce
Suppresses the confirmation step.

DELETING INTERESTING VERSIONS. *Default:* **rmver** does not delete a version to which a version label, attribute, or hyperlink is attached, or at which a branch begins.

–xbr·anch
Deletes a version even if one or more branches begin there. In the process, those branches (including all their versions and subbranches) are also deleted.

–xla·bel
Deletes a version even if it has one or more version labels.

–xat·tr
Deletes a version even if it has one or more attributes.

–xhl·ink

Deletes a version even if it has one or more hyperlinks. This also destroys the hyperlink object, thus modifying the other object to which the hyperlink was attached.

CAUTION: Using this option can delete merge arrows (hyperlinks of type **Merge**) created by the **merge** command. This may destroy essential metadata.

DATA-ONLY DELETION. *Default:* **rmver** deletes both the version object in the VOB database along with associated metadata, and the corresponding data container in a source storage pool.

–dat·a

Deletes only the data for the specified version, leaving the version object, its subbranches, and its associated metadata intact. In particular, this option preserves event records and enables continued access to the configuration record of a DO version.

CAUTION: Using this option implicitly invokes the **–xbranch**, **–xlabel**, **–xattr**, and **–xhlink** options, as well. That is, the data container is deleted even if the version has a label, attribute, or hyperlink attached or has a branch sprouting from it.

SPECIFYING THE VERSIONS TO BE REMOVED. *Default:* None.

–ver·sion *version-selector*

For each *pname*, removes the version specified by *version-selector*. This option overrides both version-selection by the view and version-extended naming. See the **version_selector** reference page for syntax details.

–vra·nge *low-version high-version*

For each *pname*, removes all versions between (but not including) the two specified versions. *low-version* and *high-version* must be on the same branch, and are specified in the same way as *version-selector*.

pname ...

(Required) One or more pathnames, indicating versions to be removed:

- A standard or view-extended pathname to an element specifies the version in the view.
- A version-extended pathname specifies a version, independent of view.

Use **–version** or **–vrange** to override these interpretations of *pname*.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**

Overrides the default with the option you specify. See the **comments** reference page.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Delete the version of **msg.c** in the view.

  cmd-context **rmver msg.c**
  ```
  Removing these versions of "msg.c":
   /main/1
  Remove versions? [no] yes
  Removed versions of "msg.c".
  ```

- Delete version 1 on the **rel2_bugfix** branch of element **util.c**, using a version selector to specify the version, suppressing confirmation prompts.

  cmd-context **rmver -force -version /main/rel2_bugfix/1 util.c**
  ```
  Removing these versions of "util.c":
   /main/rel2_bugfix/1
  Removed versions of "util.c".
  ```

- Delete version 3 on the **main** branch of element **Makefile**, even if it has labels and/or attributes. Use a version-extended pathname to specify the version.

  cmd-context **rmver -xlabel -xattr Makefile@@/main/3**
  ```
  Removing these versions of "Makefile":
   /main/3 (has: labels, attributes)
  Remove versions? [no] yes
  Removed versions of "Makefile".
  ```

- Delete all versions between **0** and **LATEST** on the **main** branch of element **hello.c**.

  cmd-context **rmver -vrange /main/0 /main/LATEST hello.c**
  ```
  Removing these versions of "hello.c":
   /main/1
   /main/2
  Remove versions? [no]yes
  Removed versions of "hello.c".
  ```

- Delete version 2 on the **main** branch of **util.c**, even if there are one or more subbranches off that version. (The subbranches, if any, are also deleted.)

cmd-context **rmver -xbranch util.c@@/main/2**
```
Removing these versions of "util.c":
 /main/2 (has: subbranches)
Remove versions? [no]yes
Removed versions of "util.c".
```

SEE ALSO

config_ccase, **describe**, **lshistory**, **lsvtree**, **rmbranch**, **rmelem**, **rmname**, **type_manager**

rmview

Removes a view or removes view-related records from a VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only—Remove a *dynamic view* view and its related records:
  rmview [–f·orce] { –tag *dynamic-view-tag* | *dynamic-view-storage-pname* }
- Remove a *snapshot view* and its related records:
  rmview [–f·orce] { *snapshot-view-pname* | *snapshot-view-storage-pname* }
- Remove only view-related records from a VOB:
  rmview [–f·orce] [–vob *vob-selector* | **–avo·bs** | **–a·ll] –uui·d** *view-uuid*

DESCRIPTION

The **rmview** command performs different, but related, tasks:

- Removing a view and its related records from a VOB
- Removing only the view-related records from a VOB

Removing a View and Its Related Records

Use this form of the command to remove a view completely. Complete removal of a view entails:

- Removing the view-storage directory
- Removing view-related records for that view from all accessible VOBs: *checkout records*, *derived object* records (ClearCase and Attache dynamic views only)
- Killing its associated **view_server** process, if the view is currently active
- For a snapshot view, also removing recursively the snapshot view's root directory, which is the directory tree of loaded versions and *view-private objects*
- For a dynamic view, removing its entry in the viewroot directory
- Removing the view's information from the view registry

Be sure that the current working directory is not within the view storage area that you are deleting.

By default, **rmview** refuses to delete a view if any element is checked out to that view. You can override this behavior with the **–force** option.

NOTE: **rmview** does not allow you to remove your current *set view* or *working directory view* (the view in which you are executing **rmview**). However, you can remove a view (set view or working directory view) that you are currently using if you issue the **rmview** command from a shell in which you are not using the view.

If the view was created with **mkview –ln**, its view-private objects are stored in a directory tree in an alternate location. **rmview** attempts to delete this directory tree; if it does not succeed, an error occurs and the view storage area remains unaffected.

Purging View-Related Records Only

Use this form of the command in either of these situations:

- Complete purging of view-related records from all VOBs is not possible. (For example, some of the VOBs may be offline when you remove the view.)
- A view storage area cannot be deleted with **rmview**, because it has become unavailable for another reason: disk crash, accidental deletion with UNIX **rm(1)**, and so on.

To remove view-related records only, use **rmview** and specify a view by its UUID (universal unique identifier; see the *View UUIDs* section). Despite being invoked as **rmview**, this form of the command has no effect on any view or **view_server** process, only on the specified VOBs.

Caution

Incorrect results occur if a VOB loses synchronization with its views. To avoid this problem:

- Never remove a view with UNIX r**m(1)**; always use the **rmview** command.
- If a view still exists, do not use **rmview –uuid** to delete records relating to it from any VOB. Make sure that the view need not be used again before using this command.

View UUIDs

Each view has a universal unique identifier. For example:

```
52000002.4ac711cb.a391.08:00:69:02:18:22
```

The listing produced by a **describe –long vob:** command includes the UUIDs of all views for which the VOB holds checkout records and derived object records.

Controlling the Size of the vista.tjf File

The file **vista.tjf** records updates to the VOB that result from **rmview** operations. **vista.tjf** can grow very large. To limit its size, read about the file **db.conf** in the **config_ccase** reference page.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the view owner or **root**. See the **permissions** reference page

Locks: No locks apply.

OPTIONS AND ARGUMENTS

CONFIRMATION STEP. *Default:* Prompt for confirmation of the specified **rmview** operation.

–f·orce

Suppresses confirmation prompts for:

- Complete view removal: confirmation is needed to proceed if some elements are checked out to the view. Proceeding has the effect of canceling the checkouts and destroying the work items: **rmview** removes the checkout records from the appropriate VOBs.
- Remove view-related records: confirmation is needed to proceed if the view still exists.

SPECIFYING A VIEW. *Default:* None.

–tag *dynamic-view-tag*

Specifies the dynamic view to be removed. *dynamic-view-tag* specifies the view-tag of a dynamic view. **rmview** removes the *view storage directory* and all relevant entries from the network's *view registry.*

dynamic-view-storage-dir-pname

Specifies the storage location directory where the dynamic view resides. Be sure that the current working directory is not anywhere within this view storage area.

snapshot-view-pname

Specifies the path to your snapshot view. This is the directory in which you load your files and do your work. **rmview** removes the *view storage directory* and all relevant entries from the network's *view registry.* Be sure that the current working directory is not anywhere within this view storage area.

snapshot-view-storage-dir-pname

NOTE: This option is intended fordeleting view storage associated with a snapshot view that was deleted using an operating system command such as rm. Only **rmview** effectively deletes a view, and in normal circumstances, you should specify *snapshot-view-pname* rather than this argument to delete a snapshot view.

Specifies the directory within a storage location where the snapshot view resides. **rmview** removes the *view storage directory* and all relevant entries from the network's *view registry.* Be sure that the current working directory is not anywhere within this view storage area.

SPECIFYING VIEW-RELATED RECORDS. *Default:* None.

–vob *vob-selector*
Specifies the VOB from which view-related records are to be removed. If you omit this option, **cleartool** or Attache uses the VOB containing the current working directory. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the *VOB-tag* (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–avobs
Specifies that view-related records are to be removed from the VOBs specified by the environment variable **CLEARCASE_AVOBS**, or if this variable is unset, from all VOBs mounted on the current host (ClearCase and Attache) or all VOBs residing on the ClearCase LT server host.

–all
Specifies that the view-related records are to be removed from all VOBs in which such records can be found.

–uuid *view-uuid*
Specifies the view whose records are to be removed from one or more VOBs.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Delete the view storage area at **/view_store/Rel2.vws**.

  cmd-context **rmview /view_store/Rel2.vws**

- Delete the view storage area whose view-tag is **anneRel2**.

  cmd-context **rmview –tag anneRel2**

- Delete the checkout and DO records for a deleted view from the current VOB. Suppress the confirmation prompt.

  cmd-context **rmview –force –uuid 249356fe.d50f11cb.a3fd.00:01:56:01:0a:4f**

  ```
  Removed references to VIEW "host2:/usr/vobstore/tut/old.vws"
     from VOB "/usr/hw".
  ```

- Remove the snapshot view **test_ssview**, even though it has checkouts.

cmd-context **rmview -tag ~usr1/test_ssview.dir**

```
VOB "/tmp/testvob" still has check-outs.
Remove view "/net/peroxide/export/home/usr1/test_ssview.dir/.view.stg"
anyway?  [no] yes
Removing references from VOB "/tmp/testvob" ...
Removed references to view
"/net/peroxide/export/home/usr1/test_ssview.dir/.view.stg" from VOB
"/tmp/testvob".
```

SEE ALSO

config_ccase, **env_ccase**, **lsview**, **mktag**, **mkview**, **registry_ccase**, **rmtag**, **unregister**

rmvob

Removes a VOB storage directory

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

rmvob [**–f**·**orce**] *vob-storage-dir-pname ...*

DESCRIPTION

The **rmvob** command deletes one or more *VOB storage directories*. Confirmation for each VOB is required, unless you use the **–force** option.

In addition to removing the VOB storage directory, **rmvob** removes all relevant entries from the network's *VOB registry.* However, **rmvob** does not unmount the VOBs. You must unmount the VOB with the **umount** command before removing a VOB storage area. (Be sure to warn users before you do.)

CAUTION: Be sure that the current working directory is not within the VOB storage area that you are deleting.

If, before using **rmvob**, you do not unmount the VOB on all hosts where it is mounted, you must use the standard operating system commands **umount** and **rmdir** to reclaim the VOB mount point on each host.

NOTE: If you remove a VOB storage area with standard UNIX commands, you must unregister the VOB with the **rmtag** and **unregister** commands.

REMOVING REPLICATED VOBS

To remove a replicated VOB, you must follow the procedure in the *ClearCase MultiSite Manual*. **rmvob** fails if the VOB replica masters any objects, unless you specify the **–force** option.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: VOB owner, or **root**. See the **permissions** reference page.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

–f·orce
Suppresses the confirmation step. If the VOB is replicated, this option allows **rmvob** to remove the VOB storage directory even if the replica masters any objects.

vob-storage-dir-pname ...
The pathnames of one or more VOB storage directories to be removed.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Unmount and delete the VOB storage area **/usr/vobstore/project.vbs** mounted on **/vobs/project**.

cmd-context **umount /vobs/project**

cmd-context **rmvob /usr/vobstore/project.vbs**

```
Remove versioned object base "/usr/vobstore/project.vbs"? [no] yes
Removed versioned object base "/usr/vobstore/project.vbs".
```

SEE ALSO

mkvob, **registry_ccase**, **umount**

rmws

Removes and unregisters a workspace

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

rmws [**–f·orce**] [*ws-name*]

DESCRIPTION

The **rmws** command removes the specified workspace and all of its local files and subdirectories.The workspace's storage directory is removed even if it is being shared with another workspace. If the associated view still exists and was created with the **mkws** command, it is removed as well. The **–force** option is applied to the **rmview** command; prompts are always issued for removal of local writable files.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE WORKSPACE. *Default:* Current workspace.

ws-name
Specifies the *workspace name* or view-tag name of the workspace to be deleted

CONFIRMATION STEP. *Default:* If the view is being deleted, **rmview** prompts for confirmation before deleting anything.

–f·orce
Automatically responds **yes** to confirmation requests that **rmview** would otherwise make:

- Deleting a *view-storage directory*: confirmation is needed to proceed if some elements are checked-out to the view. Proceeding has the effect of canceling the checkouts: **rmview** removes the *checkout records* from the appropriate VOBs.

EXAMPLES

- Remove the current workspace. At an Attache prompt:

  rmws

- Remove the workspace containing writable files, corresponding to the view with view-tag **jed_main**. At an Attache prompt:

  rmws jed_main
  ```
  \tmp\agora_hw\src\hello.c may have been modified
  \tmp\agora_hw\bin\hello.exe may have been modified
  OK to remove \jed_main? [no] yes
  Removing references from VOB "/tmp/agora_hw" . . .
  Removed references to view "/net/agora/usr/jed/views/jed_main.vws" from
  VOB "/tmp/agora_hw".
  ```

SEE ALSO

attache_command_line_interface, **attache_graphical_interface**, **mkws**, **lsws**, **rmview**

schedule

Schedules and manages jobs to be run one or more times

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

- ClearCase only—Display information about jobs, tasks, or protection:
  sched·ule [**–f·orce**] [**–hos·t** *hostname*] **–get**
  [**–sch·edule** | **–job** *job-id-or-name* | **–tas·ks** | **–acl**]
- ClearCase only—Edit a schedule or the scheduler's protection information:
  sched·ule [**–f·orce**] [**–hos·t** *hostname*] **–edi·t** [**–sch·edule** | **–acl**]
- ClearCase only—Set a schedule or protection using definitions in a file:
  sched·ule [**–f·orce**] [**–hos·t** *hostname*] **–set**
  [**–sch·edule** | **–acl**] *defn-file-pname*
- ClearCase only—Perform an operation on a scheduled job:
  sched·ule [**–f·orce**] [**–hos·t** *hostname*]
  [**–del·ete** | **–run** | **–wai·t** | **–sta·tus**] *job-id-or-name*
- ClearCase LT only—Display information about jobs, tasks, or protection:
  sched·ule [**–f·orce**] **–get** [**–sch·edule** | **–job** *job-id-or-name* | **–tas·ks** | **–acl**]
- ClearCase LT only—Edit a schedule or the scheduler's protection information:
  sched·ule [**–f·orce**] **–edi·t** [**–sch·edule** | **–acl**]
- ClearCase LT only—Set a schedule or protection using definitions in a file:
  sched·ule [**–f·orce**] **–set** [**–sch·edule** | **–acl**] *defn-file-pname*
- ClearCase LT only—Perform an operation on a scheduled job:
  sched·ule [**–f·orce**] [**–del·ete** | **–run** | **–wai·t** | **–sta·tus**] *job-id-or-name*

DESCRIPTION

The **schedule** command creates and manages ClearCase and ClearCase LT-related jobs and arranges to execute them at specified times. A job consists of an executable program, or task, that the scheduler runs one or more times with a given set of arguments.

In ClearCase, the scheduler is available on any host that runs the **albd_server**. In ClearCase LT, the scheduler is available on the ClearCase LT server host only.

Task and Job Storage

The scheduler relies on two data repositories:

- A database of tasks available for scheduling
- A database of jobs, or scheduled tasks

A task must be defined in the task database before you can schedule it. The task database is a single text file, **/var/adm/atria/scheduler/tasks/task_registry**.

root can add task definitions to the task database by editing this file using a text editor. You must not change the definitions of standard tasks, but you can add your own task definitions at the end of the file. For more information, see *Task Definition Syntax* on page 919.

Standard tasks reside in the directory *ccase-home-dir***/config/scheduler/tasks**. These tasks are not editable.

Tasks that you define can reside anywhere in the file system, but the recommended location is the directory **/var/adm/atria/scheduler/tasks**. This directory contains a task, **ccase_local_day.sh**, that is intended for user-defined operations to be run daily. The directory contains another task, **ccase_local_wk.sh**, that is intended for user-defined tasks to be run weekly. **root** can customize these two tasks using a text editor or can create entirely new tasks.

The database of jobs is the file **/var/adm/atria/scheduler/db**. This is a binary file that you can read and edit only by using the **schedule** command. When you use the **schedule** command to change the job database, you use the job definition language described in *Job Definition Syntax* on page 915.

Task and Job Database Initialization

ClearCase and ClearCase LT install a template for an initial task database, containing definitions for standard tasks, as the file *ccase-home-dir***/config/scheduler/tasks/templates/task_registry**. The **albd_server** uses this template to create the first version of the actual task database, **/var/adm/atria/scheduler/tasks/task_registry**.

Templates are installed for two customized tasks, **ccase_local_day.sh** and **ccase_local_wk.sh**, in the directory *ccase-home-dir***/config/scheduler/tasks/templates**. The **albd_server** uses these templates to create initial versions of these tasks in the directory **/var/adm/atria/scheduler/tasks**.

ClearCase and ClearCase LT install an initial set of job definitions as the text file *ccase-home-dir***/config/scheduler/initial_schedule**. These job definitions rely on task definitions in the task registry template. The **albd_server** uses these job definitions to create the first version of the job database, **/var/adm/atria/scheduler/db**.

NOTE: Do not edit or delete any files in the directory tree whose root is *ccase-home-dir*/**config/scheduler**.

Default Schedule

When no job database exists, the **albd_server** uses the initial set of job definitions in the file *ccase-home-dir*/**config/scheduler/initial_schedule** to create a default schedule. This schedule consists of some jobs run daily and other jobs run weekly.

Daily jobs:

- Scrub cleartext and derived object storage pools of all local VOBs, using **scrubber**.
- Copy the VOB database for all local VOBs that are configured for snapshots, using **vob_snapshot**.
- Copy the ClearCase registry from the primary registry server host (when run on a backup registry server host), using **rgy_backup**.
- Run user-defined daily operations in **/var/adm/atria/scheduler/tasks/ccase_local_day.sh**.
- Generate and cache data on disk space used by all local views, using **space**.
- Generate and cache data on disk space used by all local VOBs, using **space**.

Weekly jobs:

- Scrub some logs (see **errorlogs_ccase**).
- Scrub the databases of all local VOBs, using **vob_scrubber**.
- Run user-defined weekly operations in **/var/adm/atria/scheduler/tasks/ccase_local_wk.sh**.
- Generate and cache data on disk space used by derived objects in all local VOBs, using **dospace**.

The default schedule also includes three jobs to automate the synchronization of MultiSite replicas. These jobs are designed to run daily but are disabled by default, whether or not MultiSite is installed. For more information on these jobs and how to enable them for use with MultiSite, see *ClearCase MultiSite Manual*.

Job Timing Options

You can arrange for a job to run under a variety of schedules:

- Run daily or every *n* days, starting at a specified time of day and possibly repeating at a specified time interval during the day.
- Run weekly or every *n* weeks, on one or more days of each week, starting at a specified time of day and possibly repeating at a specified time interval during the day.
- Run monthly or every *n* months, on a specified day of the month, starting at a specified time of day and possibly repeating at a specified time interval during the day.
- Run immediately after another job finishes.

For daily, weekly, and monthly schedules, you can specify starting and ending dates for the job. To run a job one time, you can specify a daily schedule with identical start and end dates.

Job Definition Syntax

The **–edit** and **–set** options create or modify jobs using a declarative job definition language. The **–get** option displays a textual representation of currently defined jobs using the same language.

The job definition language has the following general features:

- Each statement must occupy a single line, though job descriptions and output messages can occupy more than one line.
- The language is case insensitive.
- Leading white space, lines beginning with a number sign (**#**), and blank lines are ignored, except within job descriptions.
- The quotation character is double quote (").

A job definition file consists of a sequence of job definitions. Each job definition begins with the statement **Job.Begin** and ends with the statement **Job.End**. Between these statements are other statements that define job properties. A statement that defines a job property has the following form:

Job.*property_name***:** *value*

Some properties have fields. In this case the definition of a property consists of a sequence of statements, one for each field, with the following form:

Job.*property_name***.***field***:** *value*

Some fields themselves have subfields.

The *value* portion of some property definitions can contain a sequence of individual values separated by commas. No white space can appear before or after a comma that separates two values in a sequence. For the **Args** property, individual values are separated by white space.

Job properties are of two types:

- Editable. You can define or modify the property. Some properties and fields are required; others are optional and have default values. When you define or modify a property, you must specify fields and subfields of that property in the order listed in Table 9 and Table 10.
- Read-only. The scheduler defines the property, and you cannot define or modify it. When you create a job definition, the scheduler ignores all definitions of read-only properties. When you edit an existing job definition, the scheduler ignores all definitions of read-only properties except for **Id**. When you edit an existing job definition, the scheduler uses the **Id**, if present (and if not present, the **Name**), to identify the job to modify.

Table 9 lists editable job properties.

Table 9 Editable Job Properties

Property	Field	Value	Default
Name		*name_string* (quoted if it contains white space; must be unique across jobs)	No default; a value is required.
Description	**Begin**	*desc_string* (on subsequent lines only; maximum 255 characters)	""
	End	(none)	
Schedule	(see Table 10)	(see Table 10)	No default; a value is required.
Task		*task_id* (unsigned) | *task_name* (string)	No default; a value is required.
Args		*arg_string* [...] (*arg_string* quoted if it contains white space)	No args
DeleteWhenCompleted		**TRUE** | **FALSE**	**FALSE**
NotifyInfo	**OnEvents**	**JobBegin** | **JobEndOK** | **JobEndOKWithMsgs** | **JobEndFail** | **JobDeleted** | **JobModified** [,...]	If no **NotifyInfo** field is specified, no notifications are issued; if any **NotifyInfo** field is specified, all must be specified.
	Using	**email**	
	Recipients	*address* [,...]	

Table 10 lists fields of the **Schedule** property. Schedules are of two types:

- Periodic. The job runs on one or more days specified by the **Monthly**, **Weekly**, or **Daily** field.

- Sequential. The job runs following completion of another job specified by the **Sequential** field.

The **Monthly**, **Weekly**, **Daily**, and **Sequential** fields are mutually exclusive; each job must have one and only one of these fields.

The **StartDate**, **LastDate**, **FirstStartTime**, **StartTimeRestartFrequency**, and **LastStartTime** fields are optional. One or more of these fields can appear along with a **Monthly**, **Weekly**, or **Daily** field. **StartDate** and **LastDate** determine the first and last dates the job is eligible to run on its monthly, weekly, or daily schedule. **FirstStartTime** determines what time the job first runs on each day it is scheduled. **StartTimeRestartFrequency** is the time interval between subsequent invocations of the job, if any, on each day it is scheduled. **LastStartTime** is meaningful only with **StartTimeRestartFrequency**; it determines the last time the job is eligible to run on each day it is scheduled.

All dates and times are local to the host on which the scheduler is running.

Table 10 Fields of the Job Schedule Property

Schedule Field	Subfield	Value	Default
Monthly	**Frequency**	*every_n_months* (unsigned)	No default; if any **Monthly** subfield is specified, all must be specified.
	Day	*day_number* | *ordinal_spec day_spec* (*day_number* ::= **1** ... **31**) (*ordinal_spec* ::= **First** | **Second** | **Third** | **Fourth** | **Last**) (*day_spec* ::= **Mon** | **Tue** | **Wed** | **Thu** | **Fri** | **Sat** | **Sun** | **Weekday** | **Weekendday** | **Day**)	
Weekly	**Frequency**	*every_n_weeks* (unsigned)	No default; if any **Weekly** subfield is specified, all must be specified.
	Days	**Mon** | **Tue** | **Wed** | **Thu** | **Fri** | **Sat** | **Sun** [,...]	
Daily	**Frequency**	*every_n_days* (unsigned)	No default
StartDate		[*d*]*d*–*month*–[*yy*]*yy* (*month* ::= **January** ... **December** | **Jan** ... **Dec**)	Today
LastDate		**StartDate** | [*d*]*d*–*month*–[*yy*]*yy* (*month* ::= **January** ... **December** | **Jan** ... **Dec**)	No last date

Table 10 Fields of the Job Schedule Property

Schedule Field	Subfield	Value	Default
FirstStartTime		[*h*]*h*:[*m*]*m*:[*s*]*s* (24-hour format)	Now
StartTimeRestartFrequency		[*h*]*h*:[*m*]*m*:[*s*]*s* (24-hour format)	No restart
LastStartTime		[*h*]*h*:[*m*]*m*:[*s*]*s* (24-hour format)	Midnight
Sequential	**FollowsJob**	*job_id* (unsigned) | *job_name* (string)	No default

Table 11 lists read-only job properties. For the **LastCompletionInfo** property, **ExitStatus** is the value returned by the **wait()** system call on UNIX or by the **GetExitCodeProcess()** function on Windows. Only the first 511 bytes of standard output and error messages are displayed.

Table 11 Read-Only Job Properties

Property	Field	Value
Id		*job_id* (unsigned)
Predefined		**TRUE** | **FALSE**
Created		*dd–month–yy.hh:mm:ss* **by** *user.group@host*
LastModified		*dd–month–yy.hh:mm:ss* **by** *user.group@host*
NextRunTime		*dd–month–yy.hh:mm:ss*
RunningStatus	**ProcessId**	*process_id* (unsigned)
	Started	*dd–month–yy.hh:mm:ss*
LastCompletionInfo	**ProcessId**	*process_id* (unsigned)
	Started	*dd–month–yy.hh:mm:ss*
	Ended	*dd–month–yy.hh:mm:ss*
	ExitStatus	*exit_status* (hexadecimal)
	Begin	*output_and_error_messages* (on subsequent lines only)
	End	(None)

Following is an example definition you can use with the **–edit** or **–set** option to create a job scheduled to run daily:

```
Job.Begin
    Job.Name: "Daily VOB Pool Scrubbing"
    Job.Description.Begin:
Scrub the cleartext and derived object storage pools of all local VOBs.
    Job.Description.End:
    Job.Schedule.Daily.Frequency: 1
    Job.Schedule.StartDate: 11-Mar-99
    Job.Schedule.FirstStartTime: 04:30:00
    Job.Task: "VOB Pool Scrubber"
Job.End
```

Following is an example definition the scheduler could display with the **–get** option for a job scheduled to run sequentially, including job properties defined by the scheduler:

```
Job.Begin
    Job.Id: 8
    Job.Name: "Weekly VOB Database Scrubbing"
    Job.Description.Begin:
Scrub the VOB database of all local VOBs.
    Job.Description.End:
    Job.Schedule.Sequential.FollowsJob: 7
    # Job.Schedule.Sequential.FollowsJob: "Weekly MVFS Log Scrubbing"
    Job.DeleteWhenCompleted: FALSE
    Job.Task: 4
    # Job.Task: "VOB DB Scrubber"
    Job.Args:
    Job.Created: 11-Mar-99.14:12:59 by fran@acme
    Job.LastModified: 11-Mar-99.14:12:59 by fred@acme
    Job.LastCompletionInfo.ProcessId: 394
    Job.LastCompletionInfo.Started: 21-Mar-99.00:30:08
    Job.LastCompletionInfo.Ended: 21-Mar-99.00:31:08
    Job.LastCompletionInfo.ExitStatus: 0x0
Job.End
```

Task Definition Syntax

A task must be defined in the task database before you can schedule the task. The task database is a text file, which you can edit using a text editor. The task database contains definitions that use a declarative task definition language similar to the job definition language.

The task definition language has the following general features:

- Each statement must occupy a single line.
- The language is case insensitive.

- Leading white space, lines beginning with a number sign (**#**), and blank lines are ignored.
- The quotation character is double quote (**"**).

The task database file consists of a sequence of task definitions. Each task definition begins with the statement **Task.Begin** and ends with the statement **Task.End**. Between these statements are other statements that define task properties. A statement that defines a task property has the following form:

Task.*property_name***:** *value*

In the task database, definitions of standard tasks appear first. You must not change or delete any of these definitions. You can add task definitions of your own at the end of the task database file.

Table 12 lists task properties.

Table 12 Task Properties

Property	Value
Id	*task_id* (unsigned; must be unique across tasks; for user-defined tasks, must be **100** or greater)
Name	*name_string* (quoted if it contains white space; must be unique across tasks)
Pathname	*pathname_string* (quoted if it contains white space)
UIInfo	*info_string* (private to ClearCase and ClearCase LT)

The scheduler uses the task **Id** property in a job definition to identify the task to run. If any scheduled jobs use a task **Id**, you must be careful not to change the task's **Id** property in the task database without also changing all references to that property in the database of scheduled jobs.

The **Pathname** value is the pathname of the executable to be invoked when the task is run. The pathname can be relative or absolute. If it is relative, the scheduler looks first for the task in *ccase-home-dir***/config/scheduler/tasks** and then in **/var/adm/atria/scheduler/tasks**.

The optional **UIInfo** property describes the task's command-line interface, such as the types of arguments the task can take. This property is used internally by ClearCase and ClearCase LT; do not specify it for a user-defined task.

Following is an example read-only definition for a standard task:

```
Task.Begin
    Task.Id:       2
    Task.Name:     "View Space"
    Task.Pathname: view_space.sh
    Task.UIInfo:   "view-spec"
Task.End
```

Following is an example definition for a user-defined task:

```
Task.Begin
    Task.Id:       100
    Task.Name:     "Daily Local Tasks"
    Task.Pathname: ccase_local_day.sh
Task.End
```

Job Execution Environment

Each task runs in a separate process started by the **albd_server**. A task has the following execution environment:

- The user identity of the task is the same as that of the **albd_server** (**root**).
- The standard input stream is closed.
- Standard output and error messages are redirected to a file and captured by the scheduler as part of the job's **LastCompletionInfo** property.
- The current directory is undefined.
- Environment variables are those in effect for the **albd_server**.

PERMISSIONS AND LOCKS

The scheduler maintains a single access control list (ACL). The ACL determines who is allowed access to the scheduler and to the ACL itself.

The **–edit –acl** and **–set –acl** options modify the ACL using a declarative ACL definition language. The **–get –acl** option displays the current ACL.

The ACL definition consists of a sequence of ACL entries. Each ACL entry must occupy a single line. Leading white space, lines beginning with number sign (**#**), and blank lines are ignored. Each ACL entry has the following form:

*identity_type***:***identity access_type*

Table 13 lists the identity types and identities allowed in ACL entries. The identity types are case insensitive.

Table 13 Identity Types and Identities in Scheduler ACL Entries

Identity Type	Identity
Everyone	(None)
Domain	*domain_name*
Group	*domain_name*/*group_name* | *domain_name**group_name*
User	*domain_name*/*user_name* | *domain_name**user_name*

In the *identity* portion of an ACL entry, the *domain_name* is an NIS domain for UNIX clients of the scheduler and a Windows NT Server domain for Windows clients of the scheduler. Note that you must supply a domain in the *identity* portion of a **Group** or **User** ACL entry. For an ACL entry with a Windows NT Server domain, *group_name* must be a global group, and *user_name* must be a domain user account. Names of domains, groups, and users are case insensitive for the scheduler.

Note that no white space can appear anywhere in an *identity_type*:*identity* specification.

Table 14 lists the access types allowed in ACL entries. The access types are case insensitive.

Table 14 Access Types in Scheduler ACL Entries

Access Type	Access to Schedule	Access to ACL
Read	Read only	Read only
Change	Read and write; can start jobs	Read only
Full	Read and write; can start jobs	Read and write

Each combination of domain and group or of domain and user represents a single identity. (Note that NIS domains differ from Windows NT Server domains, so a group or user in an NIS domain represents a different identity from the same group or user in a Windows NT Server domain.) Each single identity can have only one access type. However, access rights are inherited from **Everyone** to **Domain** to **Group** to **User** in such a way that each user has the least restrictive of all these access rights that apply to that user. For example, if a user's ACL entry specifies **Read** access but the ACL entry for the user's group specifies **Change** access, the user has **Change** access. The order of ACL entries is not significant.

root always has **Full** access to the scheduler on the local host (the computer where that user is logged on). Access rights of **root** to a scheduler on a remote host are determined by the scheduler's ACL. The default ACL is as follows:

```
Everyone: Read
```

This means that by default, everyone can read the schedule, but only **root** logged on to the computer where the scheduler is running can modify the schedule or the ACL.

Following is an example ACL definition:

```
# NIS domain acme.com
Domain:acme.com Read
# Windows NT Server domain acme
Domain:acme Read
Group:acme/users Change
User:acme.com/fran Full
User:acme/fran Full
```

OPTIONS AND ARGUMENTS

Specifying the Host

–hos·t *hostname*
Specifies the host whose schedule the command operates on. The ClearCase default is the local host. The ClearCase LT default is the ClearCase LT server host.

Disabling Prompts for Confirmation

–f·orce
Suppresses all prompts to confirm changes. By default, the command asks for confirmation before changing a schedule or ACL.

Displaying Information about Jobs, Tasks, or ACL

–get [**–sch·edule**]
Displays currently scheduled jobs using the scheduler's job definition language. For more information, see *Job Definition Syntax* on page 915. This is the default action for the **–get** option.

–get –job *job-id-or-name*
Displays the currently scheduled job identified by *job-id-or-name*, which is either a number representing the job-ID or a string representing the job name. The job display uses the scheduler's job definition language. For more information, see *Job Definition Syntax* on page 915.

–get –tas·ks
Displays the tasks defined in the task database using the scheduler's task definition language. For more information, see *Task Definition Syntax* on page 919.

–get –acl
Displays the scheduler's access control list (ACL) using the scheduler's ACL definition language. For more information, see *PERMISSIONS AND LOCKS* on page 921.

Editing a Schedule or ACL

–edi·t [–sch·edule]
Opens a text editor containing definitions of currently scheduled jobs using the scheduler's job definition language. You can use the editor to add, delete, or modify job definitions. When you are finished, save the modified schedule and exit the text editor. The scheduler then replaces the current schedule with the edited version. For more information, see *Job Definition Syntax* on page 915. This is the default action for the **–edit** option.

–edi·t –acl
Opens a text editor containing current ACL entries using the scheduler's ACL definition language. You can use the editor to add, delete, or modify ACL entries. When you are finished, save the modified ACL and exit the text editor. The scheduler then replaces the current ACL with the edited version. For more information, see *PERMISSIONS AND LOCKS* on page 921.

Setting a Schedule or ACL Using Definitions in a File

–set [–sch·edule] *defn-file-pname*
Replaces all currently scheduled jobs with the jobs defined in the file whose pathname is *defn-file-pname*. The definitions in the file use the scheduler's job definition language. For more information, see *Job Definition Syntax* on page 915. This is the default action for the **–set** option.

–set –acl *defn-file-pname*
Replaces the current ACL with the ACL defined in the file whose pathname is *defn-file-pname*. The definitions in the file use the scheduler's ACL definition language. For more information, see *PERMISSIONS AND LOCKS* on page 921.

Operating on a Scheduled Job

–del·ete *job-id-or-name*
Deletes the scheduled job identified by *job-id-or-name*, which is either a number representing the job-ID or a string representing the job name.

–run *job-id-or-name*
Immediately executes the scheduled job identified by *job-id-or-name*, which is either a number representing the job-ID or a string representing the job name. The job is run in the scheduler's job execution environment. For more information, see *Job Execution Environment* on page 921.

–wai·t *job-id-or-name*
Waits for completion and displays status of the scheduled job identified by *job-id-or-name*, which is either a number representing the job-ID or a string representing the job name. This option has no effect if the job is not running.

–sta·tus *job-id-or-name*
Displays the status of the scheduled job identified by *job-id-or-name*, which is either a number representing the job-ID or a string representing the job name. Displays the most recent process-ID, start time, end time, and exit status for the job.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Display the scheduled job whose name is **"Weekly VOB Database Scrubbing"**.

```
cmd-context  schedule –get –job "Weekly VOB Database Scrubbing"
Job.Begin
    Job.Id: 8
    Job.Name: "Weekly VOB Database Scrubbing"
    Job.Description.Begin:
Scrub the VOB database of all local VOBs.
    Job.Description.End:
    Job.Schedule.Sequential.FollowsJob: 7
    # Job.Schedule.Sequential.FollowsJob: "Weekly MVFS Log Scrubbing"
    Job.DeleteWhenCompleted: FALSE
    Job.Task: 4
    # Job.Task: "VOB DB Scrubber"
    Job.Args:
    Job.Created: 11-Mar-99.14:12:59 by fran@acme
    Job.LastModified: 11-Mar-99.14:12:59 by fred@acme
    Job.LastCompletionInfo.ProcessId: 394
    Job.LastCompletionInfo.Started: 21-Mar-99.00:30:08
    Job.LastCompletionInfo.Ended: 21-Mar-99.00:31:08
    Job.LastCompletionInfo.ExitStatus: 0x0
Job.End
```

- Edit the scheduler ACL.

```
cmd-context  schedule –edit –acl
Replace the ACL? [yes]
```

- Set the schedule on host **acme1** from job definitions in the file **jobdefs.txt**.

cmd-context **schedule –host acme1 –set jobdefs.txt**
```
Replace the entire schedule? [yes]
```

- Display the status of the scheduled job whose ID is **1**.

  cmd-context **schedule –status 1**
  ```
  Job is not currently running.

      RunningJob.CompletionInfo.ProcessId: 380
      RunningJob.CompletionInfo.Started: 25-Mar-99.04:30:01
      RunningJob.CompletionInfo.Ended: 25-Mar-99.04:31:00
      RunningJob.CompletionInfo.ExitStatus: 0x0
  ```

FILES

ccase-home-dir/**config/scheduler/initial_schedule**
ccase-home-dir/**config/scheduler/tasks/templates/task_registry**
ccase-home-dir/**config/scheduler/tasks/templates/ccase_local_day.sh**
ccase-home-dir/**config/scheduler/tasks/templates/ccase_local_wk.sh**
/var/adm/atria/scheduler/db
/var/adm/atria/scheduler/tasks/task_registry
/var/adm/atria/scheduler/tasks/ccase_local_day.sh
/var/adm/atria/scheduler/tasks/ccase_local_wk.sh

SEE ALSO

albd_server, **dospace**, **errorlogs_ccase**, **rgy_backup**, **scrubber**, **space**, **vob_scrubber**, **vob_snapshot**

schemes

X Window System resources for ClearCase and ClearCase LT Graphical User Interfaces (GUIs)

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure

SYNOPSIS

ClearCase and ClearCase LT GUIs use Common Desktop Environment (CDE) settings or can read scheme files.

DESCRIPTION

Scheme files are collections of X Window System resource settings that control the geometry, colors, and fonts used by application GUIs. By default, ClearCase and ClearCase LT do not enable schemes. Instead, the GUIs use the settings that your Common Desktop Environment (CDE) specifies. However, you can enable schemes and use the ClearCase and ClearCase LT schemes mechanism by adding the following resource to your **.Xdefaults** file:

```
*useSchemes: all
```

All ClearCase and ClearCase LT GUIs except the following support the schemes mechanism:

- **cleardescribe**
- **clearhistory**

Each scheme is implemented as a separate directory. For example, the scheme **Turner** consists of four files:

Turner/palette Defines mnemonic names for colors and fonts, using a subset of standard **cpp(1)** syntax:

```
#ifndef GAMMA_1_0
#define TextForeground         #fffff
#define BasicBackground        #002e5c
#define ScrolledListBackground  #623463
.
.
.
```

Turner/Turner	Specifies resources for use by the X Toolkit widgets that make up the GUI panels. These resources can be specified absolutely, or in terms of the mnemonic names defined in the **palette** file:
	`*XmText*marginHeight: 4`
	`.`
	`.`
	`.`
	`*foreground: TextForeground`
	`*background: BasicBackground`

Turner/ClearCasepalette	Extends and/or overrides the standard palette definitions.
Turner/ClearCase	Extends and/or overrides the standard **Turner** file definitions,

The two mnemonic map declaration files are combined, as are the two resource definition files. If the same resource is specified in the standard file and the ClearCase or ClearCase LT file, the specification in the ClearCase or ClearCase LT file is used. To add your own definitions, or to replace existing ones, either edit one or both of the ClearCase and ClearCase LT-specific files, or create your own scheme files and specify a scheme file search path as described in the section *Search Path for Schemes*.

Note that the **palette** and **ClearCasePalette** files are not actually processed by **cpp**; they are processed by the ClearCase or ClearCase LT GUI itself. The resources (**Turner** and **ClearCase**) apply only to the program that reads them. They are not added to the **RESOURCE_MANAGER** property of the root window and, therefore, do not affect other X applications.

Schemes are configured at two levels:

- Your display's X resources enable scheme use and specify the name of a particular scheme.
- A search path capability supports maintenance of systemwide and personal schemes.

Resources for Schemes

The `scheme` resource specifies a scheme to be used by the ClearCase or ClearCase LT GUI. For example:

`*scheme: Turner`	*(specifies scheme for all GUI utilities)*
`xclearcase*scheme: Turner`	*(specifies scheme for xclearcase)*

If you enable scheme use but do not specify a particular scheme, the color scheme **Lascaux** or the black-and-white scheme **Willis** is used. If you do not explicitly enable schemes, the CDE resource settings are used.

Monochrome and Grayscale Schemes. A user working on a monochrome monitor gets the **Willis** scheme automatically. A user working on a grayscale monitor gets the **Print** scheme automatically. You can override these assignments with the resources `*monoScheme` and

`*grayScheme`, respectively. If you specify an alternative scheme, it must be located in the scheme search path, which is described in the following section.

Search Path for Schemes

The GUIs use a search path to find scheme directories. The default search path is **/usr/lib/X11/Schemes:/usr/atria/config/ui/Schemes**.

You can use the environment variable **SCHEMESEARCHPATH** to specify a colon-separated list of directories to be searched instead. Each entry on this list must be in the following standard X Toolkit form:

*pathname***/%T/%N%S**

The GUIs always make these substitutions:

%T → `Schemes`
%N → *scheme-name*
%S → (null)

For example, if your **SCHEMESEARCHPATH** value is

```
/netwide/config/ui/%T/%N%S:/home/gomez/%T/%N%S
```

and your **.Xdefaults** file includes the line

```
*scheme: Rembrandt
```

then the GUI reads resource schemes from these two directories:

```
/netwide/config/ui/Schemes/Rembrandt
/home/gomez/Schemes/Rembrandt
```

The GUI searches for the first instance of each scheme file (the standard map declaration file, the standard resource definition file, and the versions of the standard files), and then concatenates the files.

NOTE: If the GUI does not find a complete set of scheme files, it returns an error. Therefore, we recommend that you include the default search path in the **SCHEMESEARCHPATH** environment variable.

International Language Support. If your site uses the language resource `*xnlLanguage` to implement pathname substitutions based on national language and/or codeset, you may want to expand customized **SCHEMESEARCHPATH** entries to use one or more of these optional substitution parameters:

%L → value of `*xnlLanguage` (*language*[*_territory*][**.***codeset*])
%l → *language*
%t → *territory* (if any)
%c → *codeset* (if any)

See X Windows System Toolkit documentation for more details on constructing directory trees to store language-dependent application text files.

Platform-specific Support. On some platforms, there are specific requirements for the location of the **Schemes** directory. See *ClearCase and MultiSite Release Notes* for information on platform-specific requirements.

FILES

ccase-home-dir/**config/ui/Schemes/Gainsborough/***
ccase-home-dir/**config/ui/Schemes/Lascaux/***
ccase-home-dir/**config/ui/Schemes/Leonardo/***
ccase-home-dir/**config/ui/Schemes/Monet/***
ccase-home-dir/**config/ui/Schemes/Print/***
ccase-home-dir/**config/ui/Schemes/Rembrandt/***
ccase-home-dir/**config/ui/Schemes/Sargent/***
ccase-home-dir/**config/ui/Schemes/Titian/***
ccase-home-dir/**config/ui/Schemes/Turner/***
ccase-home-dir/**config/ui/Schemes/VanGogh/***
ccase-home-dir/**config/ui/Schemes/Whistler/***
ccase-home-dir/**config/ui/Schemes/Willis/***

SEE ALSO

X Toolkit documentation, Common Desktop Environment documentation

scrubber

Removes data containers from VOB storage pools and removes DOs from VOB database

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

ccase-home-dir/**etc/scrubber** [**–e** | **–f** | **–o**] [**–p** *pool*[,...] | **–k** *kind*[,...]]
[**–a** | *vob-storage-dir-pname* ...]

DESCRIPTION

The **scrubber** program deletes (*scrubs*) *data container* files from the cleartext storage pools and *derived object* (DO) storage pools of one or more VOBs. It also deletes corresponding (DOs) from a VOB database. Only cleartext pools and DO pools are affected; scrubbing is not defined for source pools.

NOTE: DOs are associated with *dynamic view*s only; they are not applicable to *snapshot view*s.

Scrubbing Algorithms

scrubber implements the following scrubbing algorithms:

- Heuristic scrubbing

  By default or with the **–o** option, **scrubber** uses a free-space-analysis heuristic: it compares the current free-space level of a disk partition with a lower limit computed during its previous execution. This lower limit is stored in file **/var/adm/atria/cache/scrubber_fs_info**.

  - If the free-space level is still above the computed limit, **scrubber** does no scrubbing in that partition, regardless of the state of the storage pools within it. This performance optimization allows a quick check to take place frequently (for example, once an hour), without much system overhead.
  - If the free-space level has fallen below the limit, **scrubber** performs parameter-driven scrubbing of each storage pool in the partition.

- Parameter-driven scrubbing

  With the **–f** option, **scrubber** removes data container files from a storage pool according to the pool's scrubbing parameter settings. (The heuristic scrubbing algorithm can also fall through to this algorithm.)

When a derived object pool or cleartext pool is created with **mkvob** or **mkpool**, its scrubbing parameters are set to user-specified or default values:

maximum size	Maximum pool size (specified in KB; *default*=0)
reclaim size	Size to which **scrubber** attempts to reduce the pool (specified in KB; *default*=0)
age	Threshold to prevent premature scrubbing of recently referenced objects (specified in hours; *default*=96)

Parameter-driven scrubbing proceeds as follows:

a. Files are removed from a pool only if its current size exceeds its *maximum size* setting. In this case, **scrubber** begins deleting data containers that have not been referenced within *age* hours, proceeding on a least-recently-referenced basis.

b. The data container for a derived object is deleted only if the DO's reference count is zero. In this case, the derived object in the VOB database is deleted, too. The associated configuration record is also deleted if no other derived object is associated with it.

c. Cleartext data containers do not have reference counts; they are deleted solely on the basis of recent use.

d. Scrubbing stops when the pool's size falls below its reclaim size setting. But in no case does **scrubber** delete any object that has been referenced within the last *age* hours.

A maximum size of zero is a special case: it instructs **scrubber** to delete all data containers that have not been referenced within *age* hours, regardless of the reclaim size setting.

NOTE: The **scrubber** considers access time rather than modification time. If your backup utility changes the access time on objects, **scrubber** does not delete the object if the backup utility ran within the period of time specified by *age*.

- Everything-goes scrubbing

  With the **–e** option, **scrubber** ignores a pool's scrubbing parameters, and deletes these files:

  - All files from each cleartext pool
  - All files with zero reference counts from each derived object pool

  To avoid deleting files that are currently being used, **scrubber** does not delete any file that has been accessed in the preceding two minutes.

Automatic Scrubbing

By default, the scheduler runs **scrubber** periodically with the **–f** option, so that each pool is examined individually. See the **schedule** reference page for information on describing and changing scheduled jobs.

You can scrub one or more pools manually at any time.

Scrubber Log File

scrubber documents its work in the host's scrubber log file, **/var/adm/atria/log/scrubber_log**. For example, the following partial report describes the results of scrubbing a derived object pool.

```
04/27/99 08:03:00 Stats for VOB betelgeuse:/usr1/vobstorage/orange.vbs
Pool ddft:

04/27/99 08:03:00 Get cntr tm 918.928979
04/27/99 08:03:00 Setup tm 10631.121127
04/27/99 08:03:00 Scrub tm 1207.099240
04/27/99 08:03:00 Total tm 12757.149346
04/27/99 08:03:00 Start size 404789 Deleted 3921 Limit size 0
04/27/99 08:03:00 Start files 20349 Deleted 121 Subdir dels 0
04/27/99 08:03:00 Statistics for scrub of DO Pool ddft:
04/27/99 08:03:00 DO's 3671 Scrubs 121 Strands 1760
04/27/99 08:03:00 Lost refs 1790 No DO's 20228
04/27/99 08:03:00 No fscntrs        2
```

The first six lines, which contain elapsed times and file statistics, are included in the report for every pool. The last four lines are specific to DO pools.

`Get cntr tm`	Elapsed time for first scrubbing phase: walk the file-system tree to get pathname, size, and referenced-time information for each container in the pool.
`Setup tm`	Elapsed time for second scrubbing phase: perform setup processing specific to the kind of storage pool. For a cleartext pool, no setup is required. For a DO pool, setup is complicated; see *Processing of Derived Object Pools*.
`Scrub tm`	Elapsed time for third scrubbing phase: determine which containers to delete, and then delete them.
`Start size`	Total size (KB) of all the container files in the storage pool directory before this scrubbing.
`Deleted`	Amount of storage (KB) reclaimed by this scrubbing.
`Limit size`	Desired size of the pool (KB), as specified by the pool's *maximum size* parameter.
`Start files`	Total number of container files in the storage pool directory before this scrubbing.
`Deleted`	Number of container files deleted by this scrubbing.
`Subdir dels`	Number of empty subdirectories of the storage pool directory deleted by this scrubbing.
`DO's`	Total number of zero-reference-count DOs in the VOB database before scrubbing.

`Scrubs`	Total number of shared zero-reference-count DOs deleted by this scrubbing. (This number equals the "Deleted" count, unless the **scrubber** removed shared zero-reference-count DOs that were missing their file-system containers.)
`Strands`	Total number of *stranded DOs* deleted by this scrubbing. (These are described below.)
`Lost refs`	Total number of lost DO *reference counts* deleted by this scrubbing. (These are described below.)
`No DO's`	Total number of containers in the DO pool before scrubbing that are not associated with a zero-reference-count shared DO. (Each is presumably associated with a DO that is still referenced by some view, and hence cannot be scrubbed).
`No fscntrs`	Total number of shared zero-reference-count DOs that were missing their file-system containers. This statistic is printed only when this condition occurs; also, the **scrubber_log** displays warning messages like this one: `04/21/99 10:11:17 scrubber: Warning: Unable to remove "d/do_pool2/21/5/73f1f66679f611cea15c080009935288": No such file or directory.`

Processing of Derived Object Pools

For a DO pool, **scrubber** does more than delete old, unreferenced data containers.

- It finds and deletes all *stranded DOs* from the VOB database: DOs that were never shared, and whose data containers have been deleted from view-private storage. (The VOB database is not updated when the DO's data file is removed or overwritten in the view, due to implementation restrictions.) There are no data containers in the DO storage pool for such DOs, because they were never shared. This occurs during the second phase of scrubbing.
- It finds and deletes all lost DO *reference counts* from the VOB database. Such entries are an implementation artifact; they correspond to files that were created during a build, but deleted before the build completed. This occurs during the second phase of scrubbing.
- It deletes the derived object in the VOB database corresponding to the data container, and possibly its associated configuration record as well. This occurs during the third phase of scrubbing.
- It finds and deletes all *stranded configuration records*: CRs that do not correspond to any existing derived object.

Derived Statistics

Some interesting results can be derived from these statistics:

- Total number of derived object data containers in this pool after scrubbing:

  ```
  Start files - scrubs
  ```

- Total number of unreferenced data containers in this pool after scrubbing:

  ```
  Start files - scrubs - No DO's
  ```

- Total size (KB) of the storage pool after scrubbing:

  ```
  Start size - deleted
  ```

Controlling the Size of the vista.tjf File

The file **vista.tjf** records updates to the VOB that result from **scrubber** operations. **vista.tjf** can grow very large. To limit its size, read about the file **db.conf** in the **config_ccase** reference page.

OPTIONS AND ARGUMENTS

SPECIFYING THE SCRUBBING ALGORITHM. *Default:* Invokes the free-space-analysis heuristic described above, instead of examining pools individually.

–f
: Examines all specified pools individually, using the parameter-driven algorithm. This does not guarantee that any objects will be removed from the pools.

–e
: Examines all specified pools individually (as with **–f**), using the everything-goes scrubbing algorithm.

–o
: Same as default.

SPECIFYING THE POOLS. *Default:* All of a VOB's cleartext and derived object pools are scrubbed.

–p *pool*[,...]
: Restricts scrubbing to pools with the specified names, which may occur in multiple VOBs. The list of pool names must be comma-separated, with no white space.

–k *kind*[,...]
: Restricts scrubbing to pools of the specified kinds. Valid kinds are **do** and **cltxt**. The list of kinds must be comma-separated, with no white space.

SPECIFYING THE VOBS. *Default:* None.

–a
: Scrubs all VOBs listed in the *storage registry* whose storage directories reside on the local host. An error occurs if a VOB is listed in the registry, but cannot be found on the local host.

vob-storage-dir-pname ...
One or more pathnames of VOB storage directories, indicating the particular VOBs to be scrubbed.

EXAMPLES

- Force scrubbing of all mounted VOBs with a storage directory on the local host.

  ccase-home-dir/**etc/scrubber –f –a**

- Scrub cleartext pools in the VOB whose storage directory is **/usr/vobstore/project.vobs**, using the free-space analysis heuristic.

  ccase-home-dir/**etc/scrubber –o –k cltxt /usr/vobstore/project.vobs**

- Force scrubbing of the default derived object pool (**ddft**) and the pool named **do_staged** in all mounted VOBs with a storage directory on the local host.

  ccase-home-dir/**etc/scrubber –f –p ddft,do_staged –a**

SEE ALSO

checkvob, **config_ccase**, **schedule**, **view_scrubber**, **vob_scrubber**

setactivity

Specifies the current UCM activity for your view

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

setact·ivity [**–c·omment** *comment* | **–cfi·le** *pname* | **–cq·uery** | **–nc·omment**]
[**–vie·w** *view-tag*] { **–none** | *activity-selector* }

DESCRIPTION

The **setactivity** command sets or unsets a *current activity* for a view. The current activity is one whose change set records your current work. Each view can have no more than one current activity. When you check out an element, it is associated with the current activity.

Before resetting to another activity, the **setactivity** command checks on whether any elements of the current activity are checked out in the view and, if found, issues a warning before proceeding.

You can set an activity for a view while the activity is being delivered, but the changes made to the activity when the deliver operation is in progress are not delivered.

To clear the current activity, specify a new activity or use the **–none** option.

You cannot reset an integration activity that is in use as part of a deliver or rebase operation (nor can you clear it with **–none**).

Behavior for ClearQuest-enabled projects

When executed in a view that is associated with a ClearQuest-enabled project, this command takes an *activity-selector* that is a ClearQuest record-ID (for example, SAMPL123456) of an existing ClearQuest record. If the ClearQuest record is not already linked to an activity, the command causes an activity to be created and linked to the ClearQuest record.

When you have finished working on an activity

You can stop work on an activity in these ways:

- Deliver the activity to the project's integration stream.
- Issue another **setactivity** command, specifying a different activity selector.
- Use the **–none** option to unset the current activity in your view.

PERMISSIONS AND LOCKS

Permissions Checking: None.

Locks: An error occurs if there are locks on the following objects: UCM project VOB or the activity.

Mastership: The current replica must master the activity.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

CHOOSING A VIEW. *Default*: Current view context.

–vie·w *view-tag*
Specifies a view and stream context for the command.

SPECIFYING THE ACTIVITY. *Default:* No default.

–none
Unsets the current activity, removing it from your work area.

activity-selector
Identifies the activity to be set.

You can specify an activity as a simple name or as an object selector of the form **[activity]:***name***@***vob-selector*, where *vob-selector* specifies a project VOB (see the **cleartool** reference page). If you specify a simple name and the current directory is not a project VOB, then this command assumes the activity resides in the project VOB associated with the stream attached to the current view. If the current directory is a project VOB, then that project VOB is the context for identifying the activity.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

Unset the current activity.

cmd-context **setactivity -none**

```
Cleared current activity from view java_int.
```

- Set an activity to be the current activity.

  cmd-context **setactivity create_directories**
  `Set activity "create_directories" in view "webo_integ".`

SEE ALSO

chactivity, **lsactivity**, **mkactivity**, **rmactivity**

setcache

Changes cache settings

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- Specify the cache size for a single view:

  setcache –vie·w { **–def·ault** | **–cac·hesize** *size* } { **–cvi·ew** | *view-tag* }

- Specify the cache size for a host:

  setcache –vie·w –hos·t { **–def·ault** | **–cac·hesize** *size* }

- Specify the site-wide view cache size:

  setcache –vie·w –sit·e { **–def·ault** | **–cac·hesize** *size* }
  [**–pas·sword** *registry-password*]

- ClearCase and Attache dynamic views only—Specify MVFS cache sizes:

  setcache –mvfs { **–reg·dnc** *cnt* | **–noe·ntdnc** *cnt* | **–dir·dnc** *cnt*
  | **–vob·free** *cnt* | **–cvp·free** *cnt* | **–rpc·handles** *cnt* } ...

DESCRIPTION

The **setcache** command sets view cache sizes. Although both dynamic and snapshot views use caches, cache size is more significant for a dynamic view than for a snapshot view.

ClearCase and Attache Only—View Caches

The dynamic view caches consist mostly of data retrieved from the VOB and enable the **view_server** to respond faster to RPCs from client machines. When a **view_server** process is started, it chooses its cache size from the first of the following sources to yield a value:

- The dynamic view's cache size, which is set with **mkview –cachesize** or **setcache –view –cachesize** and stored in the file *view-storage-dir***/.view** (on the **–cache** line)
- The **view_server** host's default cache size, which is set with **setcache –view –host** and stored as a decimal number in the file **/var/adm/atria/view_cache_size**

- The site-wide cache default, which is set with **setcache –view –site** or **setsite** and stored in the site config registry
- The default value: 500 KB on 32-bit platforms, 1 MB on 64-bit platforms

NOTE: If your view uses the host value or the site-wide value and that value is changed, your view's cache size does not change until you invoke **setcache –view –default** or restart the **view_server** (with **endview –server** or a reboot).

The dynamic view cache size is allocated among the individual caches. When specifying a cache size, keep the following guidelines in mind:

- The value cannot be smaller than 50 KB for 32-bit platforms or 100 KB for 64-bit platforms.
- Do not specify a value larger than the amount of physical memory on the server host that you want to dedicate to this view.
- Values greater than approximately 4 MB do not help much in most cases.
- Verify your changes by using **getcache** to check the hit rates and utilization percentages periodically to see whether they have improved.

ClearCase and Attache Dynamic Views Only—MVFS Caches

A host's MVFS caches are used to optimize file-system performance:

- The directory name cache accelerates name translation. This cache is partitioned into three areas, each of which can be tuned with one of the **setcache –mvfs** options:
  - Directory files (**–dirdnc**)
  - Nondirectory files (**–regdnc**)
  - Names not found (ENOENT) (**–noentdnc**)

  NOTE: If processes are actively using the directory name cache, you may see the following error message when trying to resize it:

  ```
  cleartool: Error: Operation "view_mfs_set_cache_sizes" failed: Device busy.
  ```

  Ask users to stop using ClearCase actively (that is, keep their view contexts, but stop manipulating files) and enter the **setcache** command again.
- The attribute cache accelerates access to file metadata (for example, by the **stat** and **access** system calls, which are frequently called during **make** or **clearmake** operations). The **–vobfree** option sets the size of the attribute cache for VOB and view-private files that are not currently open.

- The cleartext cache accelerates the **open** system call for files in a VOB and view-private files. The **–cvpfree** option sets the size of this cache. This cache is never larger than the size of the attribute cache.
- The RPC handles cache accelerates RPCs to the dynamic view. The **–rpchandles** option sets the size of this cache; the value ought to be the maximum simultaneous number of RPCs expected from your host. If this value is too small, the **getcache –mvfs** command recommends that you adjust its size.

Values set with **setcache –mvfs** are reset when you reboot your machine. To change the values permanently, see *Administering ClearCase*.

For more information on optimizing performance, see the chapters on performance tuning in *Administering ClearCase*.

PERMISSIONS AND LOCKS

Permissions Checking:

- With **setcache –view**, you must be *root* user on the **view_server** host or the view owner.
- With **setcache –view –host** and **setcache –mvfs**, you must be *root* user.
- With **setcache –view –site**, you must know the registry password.

Locks: No locks apply. See the **permissions** reference page.

OPTIONS AND ARGUMENTS

SPECIFYING THE CACHE INFORMATION TO CHANGE. *Default:* None.

–vie·w
Sets the cache size for a single view. This immediately changes the cache size; you do not need to kill and restart the **view_server**.

–vie·w –hos·t
Sets the default cache size for the current host.

–vie·w –sit·e
Sets the site-wide default size for view caches.

–mvfs
Temporarily sets cache sizes for the MVFS. These values are reset when you reboot your machine.

SETTING THE CACHE SIZE. *Default:* None.

–def·ault
With –view: removes the **–cache** line from the **.view** file. This immediately sets the size

of the view cache to (in priority order) the host size, the site-wide size, or the default size, as described in the *DESCRIPTION* section.

With **–view –host**: deletes the **/var/adm/atria/view_cache_size** file.

With **–view –site**: removes the value for the site-wide cache from the registry.

–cac·hesize *size*
Specifies a size for the **view_server** cache. *size* must be an integer value of bytes, optionally followed by the letter **k** to specify kilobytes or **m** to specify megabytes; for example, **800k** or **3m**.

SPECIFYING THE VIEW. *Default:* None.

–cvi·ew
Sets the cache size for the current view.

view-tag
Specifies the view for which the cache size is changed.

SPECIFYING THE REGISTRY PASSWORD. *Default:* When you set the site-wide view cache size with **–view –site**, **setcache** prompts you for the registry password.

–pas·sword *registry-password*
Specifies the site-wide registry password.

SPECIFYING MVFS PARAMETERS (NOT APPLICABLE TO SNAPSHOT VIEWS). *Default:* None. You must specify at least one option. *cnt* must be an integer value; see *Administering ClearCase* for information on default and suggested values and instructions on setting the values permanently.

–reg·dnc *cnt*
Sets the number of regular file DNC entries.

–noe·ntdnc *cnt*
Sets the number of ENOENT (file not found) DNC entries.

–dir·dnc *cnt*
Sets the number of directory DNC entries.

–vob·free *cnt*
Sets the number of entries in the attribute cache.

–cvp·free *cnt*
Sets the number of entries in the cleartext cache.

–rpc·handles *cnt*
Sets the number of RPC handles cached by the MVFS.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Change the cache size for view **smg_test**.

  cmd-context **setcache –view –cachesize 800k smg_test**

```
The new view server cache limits are:
Lookup cache:                    78624 bytes
Readdir cache:                  327680 bytes
File stats cache:               137592 bytes
Object cache:                   275184 bytes
Total cache size:               819200 bytes
```

- Set the site-wide cache size.

  cmd-context **setcache –view –site –cachesize 2m**

```
Registry password: <enter registry password><ENTER>
...
```

- Set the number of RPC handles cached by the MVFS to 10 (dynamic views only).

  cmd-context **setcache –mvfs –rpchandles 10**

SEE ALSO

getcache, **mvfscache**, **setsite**, **view**, **view_server**

setcs

Sets the config spec of a view

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

setcs [**–tag** *view-tag*] { **–cur·rent** | **–def·ault** | *pname* | –**stre·am** }

DESCRIPTION

This command does not require a product license.

The **setcs** command changes the *config spec* of a view to the contents of a user-specified or system-default file, or causes the view's associated **view_server** process to flush its caches and reevaluate the current config spec. The Attache workspace is not updated to reflect any changes in the view's contents.

- For UCM views, the **setcs** command checks that the view's configuration matches the configuration defined by the stream it is attached to and, if needed, reconfigures the view. Load rules already in the view's configuration are preserved.
- (ClearCase only) If the *working directory view* differs from the *set view* (established by the **setview** command), **setcs** displays a warning message and uses the working directory view.
- In a *snapshot view*, **setcs** initiates an **update -noverwrite** operation for the current view.
- (Attache only) If the specified *file* has a corresponding local file in the workspace, it is uploaded before the remote command is executed.

See the **pwv** reference page for more on *view contexts*. See the **config_spec** reference page for a complete discussion of config specs.

Export View Config Specs

If you change the config spec of a view that is being exported for non-ClearCase access, make sure that all users who may currently have the view mounted for that purpose unmount and remount the view. Unmounting and remounting the view ensures access to the correct set of files as specified in the updated config spec.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE VIEW. *Default:* Reconfigures the current view.

–tag *view-tag*
The view-tag of any *dynamic view*; the view need not be active. To set the config spec of a snapshot view,you must be in or under the snapshot view root directory (and accordingly you do not use this option). However, you can use this option to set the config spec of a dynamic view from within a snapshot view.

SPECIFYING THE KIND OF CHANGE. *Default:* None.

–cur·rent
Causes the **view_server** to flush its caches and reevaluate the current config spec, which is stored in file **config_spec** in the view storage directory. This includes:

- Reevaluating *time rules* with nonabsolute specifications (for example, **now**, **Tuesday**)
- Reevaluating **–config** rules, possibly selecting different derived objects than previously
- Re-reading files named in **include** rules

–def·ault
Resets the view's config spec to the contents of *ccase-home-dir*/**default_config_spec**, the host's default config spec (ClearCase and ClearCase LT) or the helper host's default config spec (Attache).

pname
Specifies a text file whose contents are to become the view's new config spec.

–stre·am
For a UCM view, sets the view's config spec to that defined by the stream it is attached to. This operation preserves any load rules already in the view's config spec.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Change the config spec of the current view to the contents of file **cspec_REL3**.

cmd-context **setcs cspec_REL3**

- Change the config spec of the view whose view-tag is **jackson_vu** to the default config spec.

  cmd-context **setcs –tag jackson_vu –default**

- Have the **view_server** of the current view reread its config spec.

  cmd-context **setcs –current**

SEE ALSO

attache_command_line_interface, **catcs**, **config_spec**, **lsview**, **mktag**, **mkview**, **pwv**, **update**, **view_server**

setplevel

Changes the list of promotion levels in a UCM project VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

setplevel [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–nc**·**omment**]
[**–inv**·**ob** *vob-selector*] **–def**·**ault** *default-promotion-level promotion-level ...*

DESCRIPTION

The **setplevel** command allows you to redefine the list of baseline promotion levels for a UCM project VOB and to designate one of these levels as the default promotion level for new baselines.

Each UCM project VOB includes an ordered set of promotion levels. Promotion levels are ordered from lowest to highest and can be assigned to baselines to indicate the quality or degree of completeness of the activities and versions represented by the baseline. When a project VOB is created, it includes the following ordered set of promotion levels: **REJECTED**, **INITIAL**, **BUILT**, **TESTED**, **RELEASED**. The default promotion level is **INITIAL**. This is the level that is assigned to newly created baselines.

A baseline's promotion level is used in computing a project's list of recommended baselines. The recommended baseline for a component is the latest baseline of that component in the project's integration stream that has a promotion level greater than or equal to the project's recommended promotion level (see the **chproject** reference page).

Ordered promotion levels can be used to filter lists of baselines. Promotion level is also used to populate the default list of baselines during a rebase operation on a stream. Each project defines a default rebase level. When a project is created, the default rebase level is set to the project VOB's default promotion level. See **mkproject** and **chproject** for more information.

When you delete a level that is in use, it is not completely removed from the project VOB. Instead, its place in order is changed so that it is considered to be lower than the lowest defined level. You can list information for baselines labeled with such a promotion level **lsbl –level** command.

The promotion levels available in a VOB can be listed by running the **describe** command on the UCM project VOB object. Promotion levels can be used to filter **lsbl** output—see the **lsbl** reference page.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default*: Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE PROJECT VOB. *Default:* The project VOB containing the current working directory.

–invo·b *vob-selector*
Specifies the UCM project VOB for the project whose promotion levels are being modified.

SPECIFYING THE NEW PROMOTION LEVELS. *Default:* None.

–def·ault *default-promotion-level*
Specifies the new default promotion level. Project baselines are given the default promotion level **INITIAL** when they are created. *default-promotion-level* must be one of the specified promotion levels.

promotion-level ...
An ordered list of promotion levels that defines the promotion level set for a project VOB. List elements are ordered from lowest to highest. All elements of the set must be given.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- From the project VOB directory, modify a new project VOB's set of promotion levels by removing the **INITIAL** level and adding a **START** level. Change the default level for new baselines to **BUILT**.

  cmd-context **setplevel -default BUILT REJECTED START BUILT TESTED**

- Replace the promotion level **UNIT_TEST** with **U_TEST**.

  a. Add the new level to the current set of promotion levels:

  cmd-context **setplevel -default NEW NEW BUILT UNIT_TEST U_TEST**

  b. Find baselines that use the old promotion level:

  cmd-context **lsbl -level UNIT_TEST mybaseline**

c. Change the promotion level from UNIT_TEST to U_TEST:

cmd-context **chbl -level U_TEST the-baselines-listed-by-step-b .**

d. Remove the obsolete promotion level from the project VOB:

cmd-context **setplevel -default NEW NEW BUILT U_TEST**

SEE ALSO

chbl, chproject, describe, lsbl, mkproject

setsite

Sets or unsets site-wide properties in the site config registry

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

SYNOPSIS

- Set a site-wide property:

  setsite [**–pas·sword** *registry-password*] *property-name*=*value* ...

- Unset a property:

  setsite [**–pas·sword** *registry-password*] *property-name*= ...

DESCRIPTION

The site config registry contains site-wide properties for ClearCase and ClearCase LT. ClearCase and ClearCase LT use the value for a site-wide property when you perform an operation that uses that property and you don't specify the property's value. For example, when you create a view and do not specify one of the shareable DOs options, ClearCase uses the site-wide value.

If you don't set a site-wide property in the registry, or you unset a property, the property's default value is used. To list the properties you can set and their default values, use the **lssite –inquire** command.

You can set the following properties in the registry:

view_cache_size=*value*

When a **view_server** process is started and cannot find a cache size associated with the view or the view host, it uses the value of **view_cache_size**.

value must be an integer value of bytes.

view_interop_text_mode=*value*
When a user creates a view through the Windows GUI and does not specify the text mode, the value of **view_interop_text_mode** is used.

value must be **TRUE** (equivalent to **–tmode insert_cr**) or **FALSE** (equivalent to **–tmode transparent**).

NOTE: The value set for this property does not affect views created on UNIX machines or through the MSDOS command line.

view_shareable_dos=*value*
When a user creates a view and does not specify one of the options **–nshareable_dos** or **–shareable_dos**, ClearCase uses the value of **view_shareable_dos**.

value must be either **TRUE** or **FALSE**.

NOTE: Changing the site-wide property for shareable DOs does not change the property for existing views. To change an existing view's property, use the **chview** command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE REGISTRY PASSWORD. *Default:* **setsite** prompts you for the registry password.

–pas·sword *registry-password*
Specifies the site-wide registry password.

SETTING A PROPERTY'S VALUE. *Default*: None.

property-name=*value*
Sets *property-name* in the registry.

property-name=
Unsets *property-name* in the registry.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Set the site-wide view cache size to 2 MB.

  cmd-context **setsite –password p5d82xy9 view_cache_size=2m**
  ```
  Set site-wide default view_cache_size=2m.
  ```

- Set the site-wide view cache size to 4 MB, and set the site-wide value for DOs to nonshareable.

  cmd-context **setsite view_cache_size=4m view_shareable_dos=FALSE**
  ```
  Registry password: p5d82xy9
  Set site-wide default view_cache_size=4m.
  Set site-wide default view_shareable_dos=FALSE.
  ```

- Unset the site-wide value for shareable DOs.

  cmd-context **setsite –password 9yx28d5p view_shareable_dos=**
  ```
  Unset site-wide default view_shareable_dos (was 'FALSE')
  ```

SEE ALSO

lssite, **setcache**

ClearCase MultiSite Manual

setview

Creates a process that is set to a dynamic view

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

setview [**–log·in**] [**–exe·c** *cmd-invocation*] *view-tag*

DESCRIPTION

This command does not require a product license; also, it does not apply to snapshot views.

The **setview** command creates a process that is set to the specified *dynamic view*. The new process is said to have a *set view* context. If you specify an inactive dynamic view—one whose *view-tag* does not appear in the local host's *viewroot directory*, */view*—a **startview** command is invoked implicitly to *activate* that view.

After you set the dynamic view, you can take advantage of *transparency*: the ability to use standard pathnames to access version-controlled objects. The associated **view_server** process resolves a standard pathname to an element into a reference to one of the element's versions. See the **pathnames_ccase** reference page for further details.

Using setview in Interactive Mode

The shell command **setview** creates a subprocess. If you enter the **setview** command in interactive mode (at the **cleartool** prompt), the new dynamic view is set in the current process. To push to a subprocess of an interactive **cleartool** process, use **setview –exec cleartool**.

Whether or not you have set a dynamic view , a view-extended pathname is interpreted with respect to the explicitly named dynamic view. For example, **/view/bugfix/usr/project/foo.c** always specifies the version of element **foo.c** selected by the view **bugfix**.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply. See the **permissions** reference page.

OPTIONS AND ARGUMENTS

SHELL STARTUP PROCESSING. *Default:* Reads your **.cshrc** file, but does not read any shell login startup files when starting a shell process.

–login

Reads in your shell startup file. No error occurs if this file is missing. Use this option to gain access to your personal aliases, environment variable settings, and so on.

COMMAND TO EXECUTE IN VIEW CONTEXT. *Default:* A shell process is started, as indicated by your **SHELL** environment variable; a Bourne shell (**/bin/sh**) is started if **SHELL** has a null value or is undefined. The shell runs interactively until you exit from it.

–exe·c *cmd-invocation*

Starts a shell process and invokes the specified command line in the dynamic view specified by *view-tag*. This command inherits the environment of the shell process.

SPECIFYING THE VIEW. *Default:* None.

view-tag

Any view-tag specifying a dynamic view that is registered for the current network region. Use the **lsview** command to list registered view-tags.

EXAMPLES

- Create a shell process that is set to the dynamic view **jackson_fix** and run your shell startup script.

  cmd-context **setview -login jackson_fix**

- Create a subprocess that is set to the dynamic view **jackson_fix** and run a script named **/myproj/build_all.sh** in that process. Note that the command string must be enclosed in quotes.

  cmd-context **setview -exec "/myproj/build_all.sh" jackson_fix**

- Start the ClearCase graphical user interface in the dynamic view **test_vu**.

  cmd-context **setview –exec xclearcase test_vu**

SEE ALSO

cd, **endview**, **lsview**, **mktag**, **pathnames_ccase**, **pwv**, **shell**, **startview**, **view_server**

setws

Selects a workspace

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

setws *ws-name*

DESCRIPTION

The **setws** command selects a *workspace* and an associated *view*. The initial *working directory* is the *workspace root*. A username and password combination for the *workspace helper host* are required. You are prompted for this information if it has not already been requested, or previously stored using the **Login info** command on the **Options** menu.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

ws-name
Specifies the *workspace name* or the view-tag name of an existing workspace.

EXAMPLES

- List the existing workspaces and change to a different workspace. At a workspace prompt:

  lsws

```
Workspace name     Local storage directory     Server host
  jed_ws             C:\users\jo\jed_ws          agora
  jo_main            C:\users\jo\jo_main         agora
```

  setws jo_main

SEE ALSO

attache_command_line_interface, **attache_graphical_interface**, **mkws**, **lsws**

shell

Creates a subprocess to run a shell or other program

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command
MultiSite	multitool subcommand

SYNOPSIS

sh·ell | **!** [*command* [*arg* ...]]

DESCRIPTION

The **shell** command creates a subshell with the same *view context* as the current process. If the current process is set to one view, but the *working directory view* is different, **shell** uses the working directory view. (See the **pwv** reference page for more on this topic.)

The **shell** command is intended for use in **cleartool** and **multitool** interactive mode. If you are using single-command mode, there is no need for this command.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

PROGRAM TO RUN IN SUBPROCESS. *Default:* Runs the shell program indicated by your **SHELL** environment variable (or **/bin/sh** if your environment does not include **SHELL**). The shell runs interactively until you exit from it.

command [*arg* ...]
: Runs a noninteractive shell which, in turn, invokes the program *command*, (and, optionally, passes it one or more arguments). The subshell exits immediately after executing *command*.

EXAMPLES

- Create a subshell that is set to the same view as the **cleartool** process.

  ```
  cleartool> shell
  ```

- Create a subshell, and run a command within it.

```
% ! head -2 /etc/passwd
sysadm:*:0:0:System V Administration:/usr/admin:/bin/sh
diag:*:0:996:Hardware Diagnostics:/usr/diags:/bin/csh
```

SEE ALSO

pwv, **setview**, **csh(1)**, **sh(1)**

snapshot.conf

VOB snapshot configuration file

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure

SYNOPSIS

/var/adm/atria/config/snapshot.conf

DESCRIPTION

The file **/var/adm/atria/config/snapshot.conf** file stores information that ClearCase and ClearCase LT use to notify interested parties of VOB database snapshot activity on the local VOB host. Here are the parameters in **snapshot.conf** and their default values (which are established at installation time):

```
NOTIFICATION_PROGRAM=/usr/atria/bin/notify
NOTIFICATION_LIST=root
CONFIRMATION_ON_SUCCESS=yes
```

NOTIFICATION_PROGRAM=*email-program-pathname*

The default electronic mail program specified in the configuration file supplied with ClearCase and ClearCase LT is **/usr/atria/bin/notify**. (This program is also used if no **NOTIFICATION_PROGRAM** entry exists.) This is an architecture-specific script that invokes a native mail program.

NOTIFY_LIST=*userid*[**,**...]

A comma-separated list of user IDs to notify of VOB snapshot activity on the local VOB host. List default is a single user-ID, **root**.

CONFIRMATION_ON_SUCCESS=yes | no

Specifies whether to notify the **NOTIFY_LIST** after successful VOB snapshot operations. Default value is **yes**.

SEE ALSO

vob_restore, **vob_snapshot**, **vob_snapshot_setup**

softbench_ccase

ClearCase and ClearCase LT Encapsulation for SoftBench

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

Invoked as needed by SoftBench Broadcast Message Server

DESCRIPTION

The ClearCase and ClearCase LT Encapsulation for SoftBench enables integration of ClearCase and ClearCase LT with all of the SoftBench tools on the HP-UX 10.X, HP-UX 11, and Solaris platforms. ClearCase and ClearCase LT service and broadcast all the messages prescribed for CM systems in the document *CASE Communique: Configuration Management Operation Specifications* from the historical standard.

ClearCase and ClearCase LT add a menu to the SoftBench Development Manager, providing users with a familiar interface to the most important version control and configuration management functions. Users can customize the SoftBench environment to add items to this menu, accessing more sophisticated features.

Users can configure the SoftBench Builder to use the ClearCase build tool, **clearmake**. All other SoftBench tools (debugger, browser, static analyzer, and so on) work within ClearCase and ClearCase LT environments by using the transparent file access capability.

ClearCase and ClearCase LT can broadcast SoftBench messages whenever they perform a CM operation, no matter how that operation was requested: from the SoftBench or ClearCase or ClearCase LT graphical user interfaces, from the ClearCase or ClearCase LT command line interface, from the ClearCase API, from other SoftBench tools, and so on. This flexibility accommodates a variety of working styles without sacrificing tool integration.

SoftBench tools communicate with ClearCase and ClearCase LT through the SoftBench *Broadcast Message Server* (BMS), and two server processes:

- **clearencap_sb** — the ClearCase and ClearCase LT *encapsulator* for SoftBench
- **sb_nf_server** — the ClearCase and ClearCase LT *notice forwarder* for SoftBench

The Encapsulation supports SoftBench V5 and V6. When **clearencap_sb** or **sb_nf_server** is invoked, it automatically determines whether you are running V5 or V6 and executes the version-specific program:

V5	**clearencap_sb_prev**
	sb_nf_server_prev
V6	**clearencap_sb_curr**
	sb_nf_server_curr

NOTE: The commands **clearencap_sb –ver** and **sb_nf_server –ver** print the name of the program that will be executed (***_curr** or ***_prev**) and the version of SoftBench that is installed.

After SoftBench has been configured to work with ClearCase or ClearCase LT, certain SoftBench commands automatically invoke CM operations. When a SoftBench tool makes a configuration management request, such as `VERSION-CHECK-OUT`, the BMS receives the message and passes it on to the ClearCase/ClearCase LT encapsulator. (The BMS starts the encapsulator process if it is not already running.) The encapsulator evaluates the message and invokes the appropriate tool, such as **cleartool checkout**.

- If the operation succeeds, the encapsulator returns a message to the BMS.
- If the operation fails (that is, the tool exits with a nonzero exit status), the encapsulator returns a failure message to the BMS.

In both cases, the BMS passes the final status message back to the SoftBench tool.

You can have ClearCase/ClearCase LT tools send the success messages described above, even if the operation was not initiated by a SoftBench tool. To do so:

- Make sure that the tool and the BMS both have the environment variable **DISPLAY** set to the same value.
- Run the tool in an environment with **CLEARCASE_MSG_PROTO** set to **SoftBench**.

An error occurs in a ClearCase/ClearCase LT tool that has its **CLEARCASE_MSG_PROTO** variable set correctly, but not its **DISPLAY** variable.

NOTE: HP VUE users must add the **$ATRIAHOME/bin** directory to their search path by adding a line like the following to the file **/usr/lib/X11/vue/Vuelogin/Xconfig**:

Vuelogin*userPath:
/usr/bin/X11:/bin:/usr/bin:/etc:/usr/contrib/bin:*ccase-home-dir***/bin:/usr/lib:/usr/lib/acct**

Without this information, the encapsulator cannot find ClearCase/ClearCase LT utilities.

ENCAPSULATOR TRANSCRIPT PAD

Text output produced by encapsulator operations can be placed in a file (*results_file* in the pseudo-syntax summaries in the next section). If a result file is not specified, output is directed

to the encapsulator's dedicated transcript pad. The pad is created and appears on-screen the first time output is directed to it. The transcript pad window has a single menu, with these choices:

clear pad	Removes the current contents of the pad.
cancel	Interrupts the current encapsulator operation.
quit	Removes the transcript pad window from the screen. The transcript pad process continues to run and to collect output text. The window reappears on the next operation that sends text to the pad, with the new output appended to the existing contents of the pad.

ENCAPSULATION SUMMARY

The **clearencap_sb** program handles the SoftBench messages for the CM class listed in the pseudo-code syntax summary below. These conventions apply:

- The *context* parameter is replaced by the pathname currently selected in the SoftBench tool.
- Virtually all other parameters are optional. A default action is taken if no value is supplied for a given parameter, or if it has the string value **"-"** (except with comments, described in the following item).
- Many messages take optional comments. If a comment is not supplied, **clearencap_sb** prompts the user for a comment before acting on the message.

  NOTE: The comment string **"-"** does not indicate a default action; it is a one-character comment.
- Braces ({ ... }) indicate that a nondefault value from the message is substituted at that location.
- DEFAULT indicates that the user either did not supply the parameter or specified the string **"-"**.

Standard Messages

The following messages are specified in the historical standard:

`VERSION-CHECK-IN` *context rev options keyword comment*

```
if (keyword == "CO-LOCK")
    cleartool checkin -c comment options context
    cleartool checkout -nc context
else if (keyword == "CANCEL")
    cleartool uncheckout { options | -keep } context
else
    cleartool checkin -c comment options context
```

```
VERSION-CHECK-OUT context rev options keyword comment
    if (keyword == "CO")
        if (context{@@rev} not in current view)
                fail
        else
                succeed
    else
        cleartool checkout -c comment options context{@@rev}

VERSION-COMPARE-REVS context rev1 rev2 results_file
    if (results_file != DEFAULT && results_file != "*")
        if (rev1 == "-pred")
                cleartool diff -pred context{@@rev2} > results_file
        else
                cleartool diff context{@@rev1} context{@@rev2} > results_file
    else
        if (rev1 == "-pred")
                cleartool diff -graphical -pred context{@@rev2}
        else
                cleartool diff -graphical context{@@rev1} context{@@rev2}

VERSION-INITIALIZE context options comment
    cleartool mkelem -c comment options context

VERSION-LIST-DIR context results_file keyword options
    if (keyword == "RECURSIVE")
        cleartool ls -r options context { > results_file }
    else
        cleartool ls options context { > results_file }
```

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

```
VERSION-SET-MASTER context configuration options
    if (configuration == DEFAULT)
        cleartool setcs -default options
    else if (configuration == "*")
        cleartool edcs
    else
        cleartool setcs options configuration

VERSION-SHOW-HISTORY context results_file options
    cleartool lshistory options context { > results_file }
```

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

`VERSION-UPDATE-DIR` *context keyword options*
(no action needed with ClearCase/ClearCase LT — always succeeds)

Nonstandard Messages

The following messages are ClearCase and ClearCase LTextensions, not specified in the historical standard.

`VERSION-MAKE-DIR` *context keyword options comment*
if (*keyword* == "QUERY")
prompt for *directory-name*
cleartool mkdir -c *comment options context*[/*directory-name*]

`VERSION-MAKE-BRANCH` *context branch-type-name rev options comment*
cleartool mkbranch {-version *rev*} -c *comment options branch-type-name context*

`DERIVED-CAT-CONFIG-REC` *context do-extension results_file options*
cleartool catcr *options context*{@@*do-extension*} { > *results_file* }

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

`DERIVED-DIFF-CONFIG-REC` *context do-extension1 do-extension2 results_file options*
cleartool diffcr *options context*{@@*do-extension1*}
context{@@*do-extension2*} { > *results_file* }

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

`VERSION-MAKE-ATTRIBUTE` *context options attribute-type attribute-value comment*
if (options include "-default")
cleartool mkattr -c *comment options* -default *attribute-type context*
else
cleartool mkattr -c *comment options attribute-type attribute-value context*

`VERSION-GET-ATTRIBUTE` *context options attribute-type results_file*
cleartool describe -short *options* -aattr *attribute-type context* { > *results_file* }

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

`VERSION-MAKE-LABEL` *context options label-type comment*
cleartool mklabel -c *comment options label-type context*

`START-VIEW` *context view_tag*
cleartool startview *view_tag*

`VERSION-DESCRIBE` *context options results_file*
cleartool describe *options context* { > *results_file* }

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

`VERSION-LIST-CHECKOUTS` *context options results_file*
cleartool lscheckout *options context* { > *results_file* }

NOTE: If *results_file* is DEFAULT, output is sent to the transcript pad.

`VERSION-SHOW-VTREE` *context options results_file*
if (*results_file* = DEFAULT)
cleartool lsvtree -graphical *options context*
else
cleartool lsvtree *options context* > *results_file*

`VERSION-COMPARE-FILES` *context file2 result-file*
if (*result-file* != DEFAULT && result-file != "*")
cleartool diff *context file2* > *result-file*
else
cleartool diff -graphical *context file2*

NOTE: If *file2* is not supplied as part of the message, or is either **–** or *****, then **clearencap_sb** prompts the user for a file name (using the Motif file-selection dialog box).

`VERSION-MERGE-REVS` *context options rev*
cleartool merge -graphical *options* –to *context* –version *rev*

`DO-COMMAND` *context keyword command*
if (*command* includes the string "<context>"
first substitute *context* for this string,
then execute the resulting command
else
cleartool *command context*

NOTE: If the keyword is **DISPLAY**, output is sent to the transcript pad.

FILES

/var/adm/atria/log/ti_server_log error log for notice forwarder

SEE ALSO

ClearCase and MultiSite Release Notes

space

Reports on disk space use for views, VOBs, or file-system files or directories

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and Attache only—Report disk space used by a view or VOB:

  space { **–vie·w** | **–vob** } [**–a·ll**] [**–upd·ate**] [**–reg·ion** *network-region*]
  { **–host** *hostname* | *tag ...* }

- ClearCase LT only—Report disk space used by a view or VOB:

  space { **–vie·w** | **–vob** } [**–a·ll**] [**–upd·ate**]

- Report disk space used by file-system files or directories:

  space –dir·ectory *pname ...*

- Generate and cache data on disk space use for local views or VOBs:

  space { **–vie·w** | **–vob** } **–gen·erate** [**–scr·ub** *days*] [*tag ...*]

DESCRIPTION

The **space** command displays data on disk space use for views, VOBs, or file-system files or directories. Reports are organized by disk partition, with disk-use statistics listed both in absolute units (megabytes) and as a percentage of the capacity of the disk partition containing the storage directory.

- The report for a view includes view-private storage and administration data, as well as the space occupied by the view database. For a snapshot view, the report does not include the space occupied by the snapshot view directory tree. To display that information, use the **–directory** option and specify the root directory of the snapshot view.
- The report for a VOB includes disk use information for the *VOB database* and for each *storage pool*. Among other statistics, it includes information on backup VOB databases left behind when **reformatvob** was used.

With the **–view** or **–vob** option, **space** uses by default previously generated, cached data for a view or VOB. The **–update** option generates fresh data and updates the cache before displaying the report. With the **–directory** option, **space** does not use cached data.

The **–generate** option is intended for use by scheduled jobs. By default, the scheduler periodically runs **space** with the **–generate** option to generate and cache data on disk space use for all local views and VOBs. See the **schedule** reference page for information on describing and changing scheduled jobs.

PERMISSIONS AND LOCKS

Permissions Checking: For the **–update** option, you must have **Change** or **Full** access in the scheduler ACL on the host where each VOB storage directory resides (ClearCase and Attache), or the same access in the scheduler ACL on the ClearCase LT server host (ClearCase LT). See the **schedule** reference page.

For the **–vob –generate** option, you must be one of the following for each VOB: VOB owner, **root**. For the **–view –generate** option, you must be one of the following for each view: view owner, **root**. See the **permissions** reference page.

Locks: No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE DATA STRUCTURES. *Default:* If no **–view**, **–vob**, or **–directory** option is specified, the default is **–vob**. For the **–generate** option with no specified view or VOB tag, the default for is all local views or VOBs (ClearCase and Attache), or all views or VOBs (ClearCase LT).

–vie·w

ClearCase and Attache—Reports on one or more views, identified either as those whose storage directories reside on the host specified by **–host** or as those indicated by the specified *tag*s.

ClearCase LT—Reports on all views.

–vob

ClearCase and Attache—Reports on one or more VOBs, identified either as those whose storage directories reside on the host specified by **–host** or as those indicated by the specified *tag*s.

ClearCase LT—Reports on all VOBs.

–host *hostname*

Reports on all views or VOBs whose storage directories reside on the specified host.

tag ...

One or more tags, interpreted as view tags if you specify **–view** or as VOB-tags if you specify **–vob**. Each tag must be valid in the region specified by **–region**.

–reg·ion *network-region*
Specifies the *network region* in which each *tag* resides. The default is the region of the local host.

–dir·ectory *pname ...*
One or more pathnames, specifying files or directories in a file system.

REPORT FORMAT. *Default:* In a report on a view or VOB storage directory, storage items that are known to be small are not listed. (The contribution of these files is still included in the disk-use total.)

–a·ll
In addition to the default report, lists storage items known to be small, such as **.identity** and **.pid**.

DISPLAYING AND CACHING UP-TO-DATE DATA. *Default:* Use cached data.

–upd·ate
Computes and caches data on disk-space use at the time the command is issued, instead of using cached data, and then displays a report. The computation can take a few minutes.

GENERATING, CACHING, AND SCRUBBING DATA. *Default:* None.

–gen·erate
Computes and caches data on disk space use at the time the command is issued but does not display a report. The computation can take a few minutes. This option is intended to be used by periodic jobs run by the scheduler.

ClearCase and Attache only—The VOB or view storage directories for all specified VOBs or views must reside on the local host. If no *tag* argument is specified, the command generates data for all VOBs or views on the local host.

–scr·ub *days*
Deletes cached records of data on disk space use that are older than the specified number of *days*. A value of **–1** deletes cached records other than the one generated by the current invocation of the command, if any. Although most records are deleted, one data set per month is retained for historical purposes. This option is intended to be used in conjunction with the **–generate** option by periodic jobs run by the scheduler. The default scheduled job specifies a value of **30** for the **–scrub** option.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Report disk space use for a VOB, using cached data.

cmd-context **space –vob /projects/bigapp**

```
 Use(Mb)  %Use  Directory
   190.1    2%  source pool storage /usr1/vobs/bigapp_vob/s
   366.7    4%  derived object pool storage /usr1/vobs/bigapp_vob/d
    48.3    1%  cleartext pool storage /usr1/vobs/bigapp_vob/c
  1452.2   17%  VOB database /usr1/vobs/bigapp_vob/db
     0.0    0%  unknown item /usr1/vobs/bigapp_vob/vob_scrubber_params~
     0.1    0%  administration data /usr1/vobs/bigapp_vob/admin
    48.3    1%  cleartext pool /usr1/vobs/bigapp_vob/c/cdft
   366.7    4%  derived object pool /usr1/vobs/bigapp_vob/d/ddft
   190.1    2%  source pool /usr1/vobs/bigapp_vob/s/sdft
--------  ----  ---------------------------------------------------------
  2662.5   31%  Subtotal
  6466.2   76%  Filesystem srv1:/usr1 (capacity 8501.5 Mb)

Total usage 28-Jul-99.05:03:02 for vob "/projects/bigapp" is 2662.5 Mb
```

- Report disk space use for all views on a host, using cached data.

  cmd-context **space –view –host machine1**

```
 Use(Mb)  %Use  Directory
    16.4    7%  View private storage /export/home/fred/ccstore/fred_1/.s
     0.3    0%  View database /export/home/fred/ccstore/fred_1/db
     0.0    0%  View administration data
                  /export/home/fred/ccstore/fred_1/admin
--------  ----  ------------------------------------------------------
    16.7    7%  Subtotal
   219.1   91%  Filesystem machine1:/export/home (capacity 240.7 Mb)

 Use(Mb)  %Use  Directory
     1.5    1%  View private storage /cc/fred_2.vws/.s
     0.0    0%  View database /cc/fred_2.vws/db
     0.0    0%  View administration data /cc/fred_2.vws/admin
--------  ----  ------------------------------------------------------
     1.5    1%  Subtotal
    46.9   26%  Filesystem machine1:/ (capacity 180.5 Mb)

Total usage 27-Jul-99.04:31:39 for view "fred_1" is 16.7 Mb
Total usage 27-Jul-99.04:31:41 for view "fred_2" is 1.5 Mb
```

- Generate and cache disk space use data for a view and then display a report.

  cmd-context **space –view –update fred_1**

```
Updating space information for "fred_1" on host "machine1"
Job is running on remote host ("machine1"), waiting for it to finish.
......
Job completed successfully on remote host ("machine1").
 Use(Mb)  %Use  Directory
    17.1    7%  View private storage /export/home/fred/ccstore/fred_1/.s
     0.3    0%  View database /export/home/fred/ccstore/fred_1/db
     0.0    0%  View administration data
                  /export/home/fred/ccstore/fred_1/admin
--------  ----  ------------------------------------------------------
    17.4    7%  Subtotal
   220.6   92%  Filesystem machine1:/export/home (capacity 240.7 Mb)

Total usage 05-Aug-99.10:14:03 for view "fred_1" is 17.4 Mb
```

- Report disk space use for a file-system directory.

  cmd-context **space –directory ~sue**

```
 Use(Mb)  %Use  Directory
   191.6   80%  /net/machine1/export/home/sue
--------  ----  ------------------------------------------------------
   191.6   80%  Subtotal
   219.2   91%  Filesystem machine1:/export/home (capacity 240.7 Mb)
```

SEE ALSO

dospace, **mkview**, **mkvob**, **reformatvob**, **schedule**, **df(1M)**, **du(1M)**

startview

Starts or connects to a dynamic view's **view_server** process

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

startview *view-tag ...*

DESCRIPTION

Prerequisite: The *dynamic view* being started must already have a view-tag in the network's view-tag registry file. See the **mkview** and **mktag** reference pages.

The **startview** command enables processes on the local host to access a dynamic view, as follows:

- Establishes an RPC connection between the local host's *MVFS* (ClearCase multiversion file system) and the dynamic view's **view_server** process.
- Creates a view-tag entry in the local host's viewroot directory. If a **view_server** process is not already running, **startview** invokes one on the host where the view storage area physically resides.

The default name of the viewroot directory is **/view**. (See the **init_ccase** reference page for more information.) Thus, starting a dynamic view that has been registered with view-tag **main** creates the directory entry **/view/main**.. After this directory entry is created, any process on the local host can access the view through view-extended pathnames.

The dynamic view's view-tag must already be registered, which is accomplished either at view creation time (with a **mkview** command) or subsequently (with **mktag –view**).

NOTE: **startview** is not applicable to a *snapshot view*. To activate a snapshot view, change to the views's view-storage directory and issue a ClearCase command.

When to Use startview

Both **mkview** and **mktag** invoke **startview**. Furthermore, the **setview** command also invokes **startview**, if necessary. Therefore, it is rarely necessary to invoke **startview** explicitly. Typically, **startview** is used to establish view-extended naming access, without creating a process that is set to the view (as happens with **setview**). There are two main cases:

- Because **mkview** and **mktag** invoke **startview** on the local host only, remote users who want only view-extended naming access to the dynamic view must use **startview**.
- After your system has been stopped and restarted (see *EXAMPLES* on page 973), both local and remote users can use **startview** to reestablish view-extended naming access to a dynamic view.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE VIEW. *Default:* None.

view-tag ...
One or more currently registered view tags (that is, view tags visible to **lsview**).

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- The dynamic view **anne_Rel2** is registered, but its **view_server** process went down in a system crash. Restart **anne_Rel2**, and make it the working directory view.

  cmd-context **startview anne_Rel2**

  cd /view/anne_Rel2/usr/hw

- Create a dynamic view on the local host, and establish view-extended naming access to the view on **host3**.

  cmd-context **mkview –tag mainRel2 /view_store/mainRel2.vws**

```
Created view.
Host-local path: host2:/view-store/mainRel2.vws
Global path: /net/host2/view-store/mainRel2.vws
It has the following rights:
User : anne : rwx
Group: dev :
% rsh host3 cleartool startview mainRel2
```

  On **host3**, enter the following command:

  cmd-context **startview mainRel2**

SEE ALSO

endview, lsview, registry_ccase, setview, view_server

type_manager

Program for managing contents of element versions

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure

SYNOPSIS

- Type manager directory:

  ccase-home-dir/**lib/mgrs/manager-name**

- Methods, some or all of which are supported by each type manager:

  annotate, **compare**, **construct_version**, **create_branch**, **create_element**, **create_version**, **delete_branches_versions**, **get_cont_info**, **merge**, **xcompare**, **xmerge**

DESCRIPTION

A type manager is a suite of programs that manipulates files with a particular data format; different type managers process files with different formats. A directory type manager provides programs that compare and/or merge versions of directory elements. ClearCase and ClearCase LT provide several type managers; users can create additional ones.

Several version-control commands for file elements are implemented in two phases:

1. **Updating of the VOB database.** This phase is independent of the element's data format, and is handled directly by **cleartool**.
2. **Manipulation of the element's data.** In this phase, the data format is extremely significant, and so is handled by a particular type manager. The type manager is invoked as a separate program, rather than as a subroutine. This provides flexibility and openness, allowing users to integrate their own data-manipulation routines with ClearCase or ClearCase LT.

For example, checking in a **text_file** element involves:

- Storing information in the VOB database about who created the new version, when it was created, and so on
- Computing and storing the **delta** (incremental difference) between the new version and its predecessor.

For a different type of element—for example, a bitmap file—the delta is computed differently, or not at all, and so requires a different type manager.

TYPE MANAGERS

These are the type managers:

Type Manager	Function
whole_copy	Stores any data. Stores a whole copy of each version in a separate data container file.
z_whole_copy	Stores any data. Stores each version in a separate, compressed data container file using the **gzip** compression program. Note that compressed files generally take more time to check in (because they must be compressed), and reconstruct when first accessed (first cleartext fetch).
text_file_delta	Stores text files only (including those with multibyte text characters). Stores all versions in a single structured data container file (similar to an SCCS **s.** file or an RCS **,v** file). Uses incremental file differences to reconstruct individual versions on the fly.
z_text_file_delta	Stores text files only. Stores all versions in a single structured data container file, in compressed format using both the **gzip** compression program and deltas.
binary_delta	Stores any data. Stores each branch's versions in a separate, structured compressed data container file using **gzip**. Uses incremental file differences to reconstruct individual versions on the fly. Version deltas are determined by comparing files on a per-byte basis.
_html	Stores HTML source. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. Has its own **compare**, **xcompare**, **merge** and **xmerge** methods.
_ms_word	Stores Microsoft Word documents. Stores information and reconstructs versions in the same way as the **z_whole_copy** manager from which it is derived. On Windows, has its own **xcompare** and **xmerge** methods.
_rose	Stores Rational Rose artifacts. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. On Windows, has its own **compare**, **xcompare**, **merge**, and **xmerge** methods for which it invokes a tool specialized for Rose files.

Type Manager	Function
_xml	Stores XML source. Stores information and reconstructs versions in the same way as the **text_file_delta** manager from which it is derived. On Windows, has its own **compare**, **xcompare**, **merge**, and **xmerge** methods for which it invokes a tool specialized for XML files.
directory	Not involved in storing/retrieving directory versions, which reside in the VOB database, not in a source storage pool. This type manager compares and merges versions of the same directory element.

USING A TYPE MANAGER

To have a particular file element use a particular type manager, you must establish two connections:

```
file element ----> element type ----> type manager
```

1. Make sure the VOB has an element type that is associated with the desired type manager. Use the **lstype** command to identify an existing element type. Alternatively, use the **mkeltype –manager** command to create a new element type that is associated with the desired type manager.
2. Create the file element, specifying the element type with the **–eltype** option. If the file element already exists, use the **chtype** command to change its element type.

You can automate the assignment of the new element type to newly created elements using the file-typing facility, driven by **.magic** files. See the **cc.magic** reference page for details.

TYPE MANAGER STRUCTURE

A type manager is a collection of programs in a subdirectory of *ccase-home-dir***/lib/mgrs**; the subdirectory name is the name by which the type manager is specified with the **–manager** option in a **mkeltype** command.

Methods

Each program in a type manager subdirectory implements one **method** (data-manipulation operation). A method can be a compiled program, a shell script, or a link to an executable. It is invoked at the appropriate time by a ClearCase or ClearCase LT version-control command.

A type manager can include these methods:

create_element Invoked by **mkelem** to create an element's initial data container.

create_branch Invoked by **mkbranch** to create a branch in an element's version tree.

create_version	Invoked by **checkin** to store a new version of an element.
annotate	Invoked by **annotate** to produce an annotated listing of a version's contents.
construct_version	Invoked by a view's **view_server** process when a file element is opened, from versions stored in delta or compressed format. This method constructs a readable, *cleartext* copy of a particular version. After the cleartext version is constructed, its line terminators may be adjusted by the **view_server**, according to the view's text mode. See *Text Files, Cleartext, and a View's Text Mode* in the **mkeltype** reference page, and also **mkview**.
get_cont_info	Invoked by **checkvob** to determine the contents of a container. This method must be implemented to enable **checkvob** to fix container problems for the type manager.
delete_branches_versions	Invoked by **rmver** and **rmbranch** to delete versions of an element.
compare, **xcompare**	Invoked by **diff** to run a file-comparison program that is specific to the element's data format.
merge, **xmerge**	Invoked by **merge** to run a file-merge program that is specific to the element's data format.

A type manager need not implement every method. For example, a type manager for bitmap graphics images may omit the **merge** method, because the operation doesn't make sense for that file format. In this case, the command **cleartool merge** produces an error when invoked on an element that uses this type manager.

Method Inheritance and Links

A type manager can use symbolic links to inherit one or more of its methods from another type manager. For example, the type manager for **nroff(1)** source files described in *Example* in the section *CREATING A NEW TYPE MANAGER* is a variant of the standard **text_file_delta** type manager: it uses links to inherit all the **text_file_delta** methods, except for **compare**.

Another typical use of symbolic links is to have individual methods be links to a master type manager program, which implements several (or all) of the methods. For an example, see directory *ccase-home-dir***/lib/mgrs/z_whole_copy**.

A link to the **cleardiff** program can implement the **compare** and/or **merge** method for text files. Similarly, a link to the **xcleardiff** program can implement the **xcompare** and/or **xmerge** method. Again, see directory *ccase-home-dir***/lib/mgrs/z_whole_copy** for an example.

Data Containers

Type managers process data containers, each of which stores the actual data for one or more versions of some element. (Although growth may cause a container to split, versions never span container boundaries.) All data containers are standard UNIX files, and are stored in the VOB's source pools, which are standard UNIX directories. Only type managers deal with data containers directly; users always manipulate data using the names of elements and links.

Performing the data manipulation for a version-control operation involves several programs. For example, when ClearCase or ClearCase LT create a new version of an element:

1. The pathname (within a source pool) is generated for a new data container.
2. On the VOB host (where the VOB storage area resides), a **vob_server** process creates an empty file at that pathname.
3. On the client host (where the user is working), the type manager fills the new data container with the data for the new version. (If the type manager implements deltas, it writes the data for one or more other versions to the new container, too.)
4. The **vob_server** changes the access mode of the new data container, making it unwritable.
5. The **db_server** updates the VOB database to reference the new container.
6. Using the **MGR_DELETE_KEEP_JUST_NEW** exit status returned by the type manager, the **vob_server** deletes the old data container.

NOTE: Even with a type manager that implements deltas, a new data container is created each time a new version is created. In this case, the old container (which may have stored 27 versions) is replaced by the new container (which stores 28 versions). A type manager must never write to an old container or delete a old container (it usually does not have rights to do so).

Source Pool Data Container Names

A container leaf name includes a type manager ID to aid **checkvob** in salvaging nonreferenced containers. Here is the format of a source pool data container name (in **s/sdft**, for example):
./nn/nn/*type-mgr-id****–****orig-oid-str****–****xx***

type-mrg-id is a one-, two-, or three-character string. One-character values correspond to the predefined type managers. Two-digit values correspond to type managers with names that begin with underscore (_), and three-digit values are computed by hashing user-defined type manager names.

NOTE: Names of user-defined type managers must not begin with underscore.

CREATING A NEW TYPE MANAGER

You can create any number of new type managers for use throughout the local network. Use these guidelines:

1. Choose a name for the new type manager—ideally one that shows its relationship to the data format (for example, **bitmap_mgr**). Create a subdirectory of *ccase-home-dir***/lib/mgrs** with this name.

   NOTE: Names of user-defined type managers must not begin with underscore.
2. Create symbolic links to make the new type manager inherit some of its methods (file-manipulation operations) from an existing type manager.
3. Create your own program for the methods that you wish to customize. See *WRITING A TYPE MANAGER PROGRAM*.
4. On each other ClearCase or ClearCase LT client host in the network, either make a copy of the new type manager directory, or create a symbolic link to it. The standard storage, performance, and reliability trade-offs apply.

NOTE: An element type belongs to a VOB, and thus is available on every host that mounts its VOB. But a type manager is host-specific—it is some host's *ccase-home-dir***/lib/mgrs/manager-name** directory.

Example

This section describes how the guidelines presented above can be used to create a type manager for elements that store a project's **nroff(1)** source files. You want to implement the **compare** method with a program that compares the corresponding formatted files, instead of the source files. But you do not want to change the **merge** method, because that is an operation to be performed on the source files themselves. Thus, the type manager has these characteristics:

- It is a refinement of the **text_file_delta** type manager.
- It has the same functionality as **text_file_delta** for all methods except **compare**.
- It compares two or more versions by first creating formatted pure-ASCII files with **nroff(1)**, then using **cleardiff** to display the differences.

The example assumes that *ccase-home-dir* is the directory **/usr/atria**. Substitute your installation directory as appropriate.

The step numbers below correspond to those in the preceding section.

1. Create a directory for the type manager:

   % **su**

   <enter password>

   % **cd /usr/atria/lib/mgrs**

   % **mkdir nroff_delta**
2. For all the methods except **compare**, create symbolic links back to the **text_file_delta** directory:

```
% cd nroff_delta
% ln -s ../text_file_delta/annotate annotate
% ln -s ../text_file_delta/construct_version construct_version
% ln -s ../text_file_delta/create branch create_branch
% ln -s ../text_file_delta/create_element create_element
% ln -s ../text_file_delta/create_version create_version
% ln -s ../text_file_delta/delete_branches_versions delete_branches_versions
% ln -s ../text_file_delta/get_cont_info get_cont_info
% ln -s ../text_file_delta/merge merge
% ln -s ../text_file_delta/xcompare xcompare
% ln -s ../text_file_delta/xmerge xmerge
```

3. Create a **compare** program as an executable shell script. For example:

```
#!/bin/sh
# read file that defines methods and exit statuses
OPTS=""
while (expr $1 : '\-' > /dev/null) ; do
 OPTS="$OPTS $1"
 if [ "$1" = "$MGR_FLAG_COLUMNS" ] ; then
 shift 1
 OPTS="$OPTS $1"
 fi
 shift 1
done
COUNT=1
for X in $* ; do
 nroff -man $X | col | ul -Tcrt > /usr/tmp/compare.$$.$COUNT
 COUNT=‘expr $COUNT + 1‘
done
echo Comparing files: $*
cleardiff -quiet $OPTS /usr/tmp/compare.$$.*
rm -f /usr/tmp/compare.$$.*
exit 0
```

4. Create symbolic links to (or create copies of) the **nroff_delta** directory in other hosts' **/usr/atria/lib/mgrs** directories. For example:

```
# rlogin saturn
# cd /usr/atria/lib/mgrs
# ln -s /net/neptune/usr/atria/lib/mgrs/nroff_delta nroff_delta
```

The **nroff_delta** type manager is now ready for use.

1. Create the **manpage** element type, associating the new type manager with it:

```
% cleartool mkeltype -supertype text_file -manager nroff_delta manpage
Comments for "manpage":

Variant of text_file, for nroff source files
.
Created element type "manpage".
```

2. Convert an existing element to type **manpage**:

```
% cleartool chtype -force manpage hello.1
Changed type of element "hello.1" to "manpage".
```

3. Compare two versions of the element, using the new type manager:

```
% diff hello.1@@/main/3 hello.1
7,8c7,8
< program outputs the message
< ‘‘Hello, World’’
---
> program sends the message
> ‘‘Hello there, World’’
```

```
% cleartool diff -serial hello.1@@/main/3 hello.1
Comparing files: hello.1@@/main/3 hello.1
-----[12 changed to 12]-----
< The program outputs the message ‘‘Hello, World’’ to the
---
> The program sends the message ‘‘Hello there, World’’ to the
```

4. Revise your personal **magic** file (for example, **$HOME/.magic/my.magic**) so that certain newly created elements are assigned the **manpage** element type. For example:

```
manpage text_file : -name "*.[1-8]"
```

Classifies any file whose name ends with a digit extension (**.1**-**.8**) as a **manpage** file.

```
manpage text_file : -name "manual_pages/*"
```

Classifies any file within the **manual_pages** directory as a **manpage** file.

WRITING A TYPE MANAGER PROGRAM

When invoking a type manager method, **cleartool** passes to it all the arguments needed to perform the operation, in ASCII format. For example, many methods accept a *new_container_name* argument, specifying the pathname of a data container to which data is to be written.

In many cases, one or more of the parameters can be ignored. For example, the **create_version** method is passed **pred_container_name**, the pathname of the predecessor version's data container. If the type manager implements incremental differences, this is required information; otherwise, the predecessor's data container is of no interest.

Arguments are often object identifiers (OIDs). You need not know anything about how OIDs are generated; consider each OID to be a unique name for an element, branch, or version. In general, only type managers that store multiple versions in the same data container need be concerned with OIDs.

For more information on argument processing, see files *ccase-home-dir*/**lib/mgrs/mgr_info.h** (for C language programs) and *ccase-home-dir*/**lib/mgrs/mgr_info.sh** (for Bourne shell scripts), along with *Managing Software Projects with ClearCase*.

Exit Status of a Method

A user-defined type manager method must return an exit status to **cleartool**, indicating how the command is to be completed. The symbolic constants in *ccase-home-dir*/**lib/mgrs/mgr_info.sh** specify all valid exit statuses. For example, an invocation of **create_version** may create a new data container, then return the exit status **MGR_STORE_KEEP_JUST_NEW**; if creation of the new data container fails, it returns the exit status **MGR_STORE_KEEP_JUST_OLD**.

FILES

ccase-home-dir/**lib/mgrs/***
ccase-home-dir/**lib/mgrs/mgr_info.h**
ccase-home-dir/**lib/mgrs/mgr_info.sh**

SEE ALSO

mkelem, **mkeltype**, **cc.icon**, **cc.magic**, **gzip**

type_object

Prototype for data items stored in a VOB

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure
Attache	data structure

DESCRIPTION

A type object is a prototype for one or more data items stored in a VOB database. A user creates the data items by entering commands that create instances of the type object.

For example, to attach the version label **BASELEVEL_4.2** to the current versions of a set of source files:

- Verify that there exists a type object that defines the version label. Or, you create such a label type object (**mklbtype** command).
- Create the individual version labels as instances of the **BASELEVEL_4.2** label type object (**mklabel** command).

NOTE: The phrase "type object" is sometimes shortened to "type" in this documentation.

INSTANCES OF TYPE OBJECTS

Creating an instance of a type object is not a copy operation. Rather, the instance is a reference to the type object. In the example above, attaching version label **BASELEVEL_4.2** to a particular version does not make a copy of the **BASELEVEL_4.2** type object. Instead, it establishes a connection between the version object and the label type object.

This scheme makes it easy to administer type objects and their instances. For example, renaming the label type object from **BASELEVEL_4.2** to **BL4.2** renames all its existing instances.

NOTE: Creating an instance does not make a copy of the type object, but in certain cases it does create a new object. For example, the **mkbranch** command creates a new branch object and creates a reference connecting the new branch object to an existing branch type object. See *PREDEFINED AND USER-DEFINED TYPE OBJECTS* for information on which types are associated with objects.

KINDS OF TYPE OBJECTS

To support different kinds of data items (version labels, attributes, branches, hyperlinks, and so on), there are different kinds of type objects:

Attribute type (mnemonic: **attype**)
: Instances are attributes. The following kinds of objects can have one or more attributes attached to them: elements, branches, versions, hyperlinks, VOBs, VOB replicas, any type object.

Branch type (mnemonic: **brtype**)
: Instances are *branches*, which are objects in their own right.

Element type (mnemonic: **eltype**)
: Instances are *elements*, which are objects in their own right.

Hyperlink type (mnemonic: **hltype**)
: Instances are hyperlinks, which are objects in their own right.

Label type (mnemonic: **lbtype**)
: Instances are version labels. One or more version labels can be attached to any version object.

Trigger type (mnemonic: **trtype**)
: Instances are *triggers*.

The mnemonics listed above are useful as object-selector prefixes in ClearCase product family commands. By default, many commands consider an unprefixed name to be a reference to a file system object, typically a version of an element:

cmd-context **describe hoople**
: *(describe the file named 'hoople' in the current working directory)*

cmd-context **describe brtype:hoople**
: *(describe the branch type object named 'hoople')*

PREDEFINED AND USER-DEFINED TYPE OBJECTS

Each VOB is created with a set of predefined type objects. Users can create additional type objects.

- Predefined element types:

  file_system_object (internal only; cannot be used in a **mkelem** command)
  directory
  file
  binary_delta_file
  compressed_file

text_file
compressed_text_file

- Predefined branch type: **main**
- Predefined label types:

  CHECKEDOUT
  LATEST

- Predefined attribute types:

  HlinkFromText
  HlinkToText

- Predefined hyperlink types:

  AdminVOB
  GlobalDefinition
  Merge
  RelocationVOB

SCOPE OF TYPE OBJECTS

Each VOB has its own set of type objects, for use in creating instances of the types in that particular VOB. For example, you can attach **RLS7.0** version labels within a particular VOB only if label type **RLS7.0** already exists in that VOB. Exception: The global type object facility creates type objects on demand by copying them from a networwide repository. See *GLOBAL TYPES AND ADMINISTRATIVE VOBS* for details.

GLOBAL TYPES AND ADMINISTRATIVE VOBS

You can use the global type facility to effectively increase the scope of a type object from a single VOB to a group of VOBs—perhaps all the VOBs in your local area network. You can create any number of global type objects in one or more central administrative VOBs.

For more information about using global types, see *Administering ClearCase*.

OPERATIONS ON TYPE OBJECTS

The following commands operate on type objects. For more information about the commands, see *ClearCase Reference Manual*. For more information about how these commands work with global types, see *Administering ClearCase*.

Purpose	Commands
Type object creation	**mkattype**, **mkbrtype**, **mkeltype**, **mkhltype**, **mklbtype**, **mktrtype**
Instance creation	**mkactivity**, **mkattr**, **mkbranch**, **mkelem**, **mkhlink**, **mklabel**, **mktrigger**
Renaming	**rename**

Purpose	Commands
Copying	**cptype**
Deleting	**rmtype**
Describing/listing	**describe**, **lstype**
Lock/unlock	**lock**, **unlock**

SEE ALSO

describe, **mk**type**, **rename**, **rmtype**

umount

Deactivates a VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

umount { *vob-tag* | **–a**·**ll** }

DESCRIPTION

The **umount** command deactivates one or more VOBs on your host by unmounting them as UNIX-level file systems. A VOB is activated on a host by mounting it as a file system of type MVFS (ClearCase *multiversion file system* type). The VOB-tag by which an individual VOB is referenced is the same as the full pathname to its mount point.

umount calls the standard **umount(1M)** command.

Note that **umount** has no impact on a VOB's entries in the **vob_object** and **vob_tag** registry files.

Unmounting of Public and Private VOBs

The **root** user can unmount any VOB; other users can unmount any public VOB, and private VOBs they own.

See the **mkvob** reference page for a discussion of public and private VOBs.

Unmounting All VOBs

umount –all unmounts all public VOBs listed in the VOB registry and all private VOBs owned by the user.

CAUTION: (Except on Solaris) If you enter **umount –all** as **root** on a platform that supports the operating system command **umount –a**, the viewroot directory (**/view**) is unmounted. To remount the viewroot directory, you must stop and restart ClearCase.

PERMISSIONS AND LOCKS

Permissions Checking: See *Unmounting of Public and Private VOBs*. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

SPECIFYING THE VOB. *Default:* None.

vob-tag

Unmounts the VOB with this *VOB-tag*, which you must specify exactly as it appears in the **vob_tag** registry file.

–a·**ll**

Unmounts all public VOBs listed in the VOB registry and all private VOBs owned by the user.

EXAMPLES

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Unmount the VOB storage directory that is registered with VOB-tag **/vobs/Rel4**.

  cmd-context **umount /vobs/Rel4**

- Unmount all VOBs registered with public VOB-tags.

  % **su** *(become root user)*

  cmd-context **umount –all** *(unmount all public VOBs)*

SEE ALSO

lsvob, **mkvob**, **mount**, **register**, **registry_ccase**, **umount(1M)**

uncheckout

Cancels a checkout of an element

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

uncheck·out | **unco** [**–kee·p** | **–rm**] [**–cact**] *pname* ...

DESCRIPTION

The **uncheckout** command cancels a checkout for one or more elements, deleting the checked-out version. Any metadata (for example, attributes) that you attached to a checked-out version is lost. After you cancel a checkout:

- A *dynamic view* reverts to selecting a checked-in version of each element.
- A *snapshot view* performs an update operation for each unchecked-out element. (For snapshot views, there is an exception for the canceling of a directory checkout; see *Canceling a Directory Checkout* for more information).

In Attache, if **–rm** is not specified, any corresponding local files are uploaded before the **uncheckout** command is executed remotely, so that they can be kept in the view if **–keep** is specified or if the keep query receives a **yes** answer. Local files that correspond to canceled checkouts are updated from the version selected by the view after the checkouts are canceled, and are made read-only.

The checkout version event record for each element is removed from its VOB's database. (There is no **uncheckout** event record.) Reserve and unreserve records are also removed.

If you checked out a file under an alternate name (**checkout –out**), you cannot use the alternate name to cancel the checkout—you must use the element name listed by **ls –vob_only**.

Canceling a Checkout in an Inaccessible View

(ClearCase and ClearCase LT only) You can cancel another dynamic view's checkout by using a *view-extended pathname* to the element. For a snapshot view, or in the case where a dynamic view is no longer accessible (for example, it was deleted accidentally), a view-extended pathname does not work. Instead, do the following:

1. Enter the command **describe –long vob:***pname-in-vob*, where *pname-in-vob* is the VOB-tag of the VOB containing the checked-out file. The output of this command includes a list of views with checkouts in the VOB.
2. Look for the view-storage pathname of the inaccessible view, and note the view's unique identifier (UUID).
3. Use the uuid in the command **rmview –uuid** *uuid* to remove all of the view's checkout records from the VOB.
4. Repeat Step #3 in each VOB that may have been accessed with the view.

You can also change reserved checkouts in that view to unreserved. There is no way to cancel checkouts in an inaccessible view.

Canceling a Directory Checkout

If you cancel a directory's checkout in a dynamic view after changing its contents, the changes made with **rmname**, **mv**, and **ln** are lost. Any new elements that were created (with **mkelem** or **mkdir**) become orphaned; such elements are moved to the VOB's **lost+found** directory, stored under names of this form:

*element-name***.***UUID*

uncheckout displays a message in such cases:

```
cleartool: Warning: Object "foo.c" no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory as
 "foo.c.5f6815a0a2ce11cca54708006906af65".
```

When you use **uncheckout** in a snapshot view, the changes made with **rmname**, **mv**, and **ln** are lost in the VOB because there is no **lost+found** directory, but the snapshot view does not reflect the VOB contents until you invoke **update**.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: version creator, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch.

OPTIONS AND ARGUMENTS

HANDLING OF THE FILE. *Default:* For file elements only, **uncheckout** prompts you to decide whether to preserve a copy of the checked-out version of the element:

```
Save private copy of "util.c"? [yes]
```

A **yes** answer is equivalent to specifying the **–keep** option; a **no** answer is equivalent to specifying the **–rm** option.

–kee·p

Preserves the contents of the checked-out version (in Attache, in the view) under a file-name of the form *element-name***.keep** (or, to prevent name collisions, *element-name***.keep.1**, *element-name***.keep.2**, and so on). This file is not downloaded to the Attache workspace.

–rm

Does not preserve the contents of the checked-out version. Thus, any edits that had been made to the checked-out version are lost.

–cact

Cancels the checkout for each checked out version in the current activity.

SPECIFYING THE ELEMENT. *Default:* None.

pname ...

One or more pathnames, each of which specifies an element. The checkout in the current view is canceled, unless you use a view-extended pathname to specify another view.

NOTE: Avoid using a version-extended pathname. For example, you cannot use **hello.c@@/main/sub1** to cancel another view's checkout on the **sub1** branch of element **hello.c**.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Cancel the checkout of file element **util.c**.

  cmd-context **uncheckout util.c**

  ```
  Save private copy of "util.c"? [yes] no
  Checkout cancelled for "util.c".
  ```

- (Dynamic view only) Cancel the checkout of file **hello.h** in the **jackson_fix** view, and delete the view-private copy.

  cmd-context **uncheckout –rm /view/jackson_fix/usr/hw/src/hello.h**

  ```
  Checkout cancelled for "/view/jackson_fix/usr/hw/src/hello.h".
  ```

- Cancel the checkout of directory **subd** after creating a new element named **conv.c**. Because the context for this command is a dynamic view, the element is moved to the VOB's **lost+found** directory.

cmd-context **uncheckout subd**

```
cleartool: Warning: Object “conv.c” no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory as
 "conv.c.3d90000112fc11cba70e0800690605d8".
Checkout cancelled for "subd".
```

SEE ALSO

checkin, **checkout**, **lscheckout**, **mkview**, **reserve**, **unreserve**, **update**, **attache_command_line_interface**, **attache_graphical_interface**

unlock

Unlocks an object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

unlock [**–c**·**omment** *comment* | **–cfi**·**le** *comment-file-pname* |**–cq**·**uery** | **–cqe**·**ach** | **–nc**·**omment**]
{ [**–pna**·**me**] *pname* ... | *object-selector* ... }

DESCRIPTION

The **unlock** command removes an existing lock from an entire VOB, or from one or more objects, type objects, or VOB storage pools. See the **lock** reference page for a description of locks.

PERMISSIONS AND LOCKS

See the **lock** reference page for a description of restrictions.

OPTIONS AND ARGUMENTS

See the **lock** reference page for a description of the options to the **unlock** command.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Unlock the label types **REL1** and **REL2**.

  cmd-context **unlock lbtype:REL1 lbtype:REL2**

  ```
  Unlocked label type "REL1".
  Unlocked label type "REL2".
  ```

- Unlock the **v3_bugfix** branch.

  cmd-context **unlock cmd.h@@/main/v3_bugfix**

SEE ALSO

lock, **lshistory**, **lslock**, **protect**

unregister

Removes an entry from the **vob_object** or **view_object** registry

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- Unregister a VOB:

  unreg·ister –vob { **–uui·d** *uuid* | *vob-storage-dir-pname* }

- Unregister a view:

  unreg·ister –view { **–uui·d** *uuid* | *view-storage-dir-pname* }

DESCRIPTION

The **unregister** command removes the entry for a particular VOB or view from the network's **vob_object** or **view_object** registry. This does not affect VOB-tag or view-tag registry entries, and it does not affect the contents of the physical storage directories. See the **registry_ccase** reference page for a discussion of the registry.

If you remove a VOB or view storage directory with an operating system command (**rm –rf**, for example), instead of **rmvob** or **rmview**, the VOB or view remains unregistered. In this case, you must use the **–uuid** option to unregister the associated storage directory (and use **rmtag** to remove relevant tag entries, if any still exist).

Other Commands that Affect Storage Registries

The **mkview** and **mkvob** commands add an entry to the appropriate registry; the **rmview** and **rmvob** commands remove registry entries (and the actual storage directories as well). You can use the **register** command to update an existing entry, or to re-register a VOB or view that has been unregistered.

The **reformatvob** command updates a VOB's object registry entry (or creates one, if necessary), but does not affect its tag registry entries.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

UNREGISTERING VIEWS AND VOBS. *Default:* None.

–vob *vob-storage-dir-pname*
–vob –uuid *vob-uuid*
Use either form to specify the VOB whose **vob_object** registry entry is to be deleted. Use the VOB replica UUID reported by **lsvob –long** (not the VOB family UUID).

–view *view-storage-dir-pname*
–view –uuid *view-uuid*
Use either form to specify the view whose **view_object** registry entry is to be deleted.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

Unregister a VOB storage directory.

cmd-context **unregister –vob /vobstore/vob2.vbs**

- Unregister a view storage directory.

  cmd-context **unregister –view /view_store/view5.vws**

- Using the **–uuid** option, unregister a VOB storage directory that was deleted with **rm –rf**instead of **rmvob**. In this example, the *VOB replica UUID* (do not use the *VOB family UUID*) is found in the output from **lsvob –long**. After unregistering the storage directory, remove the VOB-tag. If the VOB has tag registry entries for more than one network region, the **–all** option removes them all.

  cmd-context **lsvob –long /vobs/src** *(find the VOB replica uuid)*

```
Tag: /vobs/src
 Global path: /net/neptune/vobstore/src.vbs

.
.
.
```

```
Vob replica uuid: cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3
```

ls /net/neptune/vobstore/src.vbs *(verify storage directory was removed)*

```
UX:ls: ERROR: Cannot access /net/neptune/vobstore/src.vbs: No such file or
directory
```

cmd-context **unregister –vob –uuid cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3**

cmd-context **rmtag –vob –all /vobs/src**

- As in the previous example, unregister a removed, but still registered, VOB storage directory. In this example, the VOB-tag has already been removed. Therefore, use the **/var/adm/atria/log/scrubber_log**, not **lsvob**, to find the VOB replica UUID. (**lsvob** lists only VOBs that have registered VOB-tags.) The **scrubber** utility, which runs nightly by default, reports the required UUID in an error message after failing to find the registered storage directory.

  % **cat /var/adm/atria/log/scrubber_log**

```
.
.
.
05/27/99 04:30:58 scrubber: Error: Unable to get VOB tag registry
information for
 replica uuid "cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3": ClearCase object
not found
05/27/99 04:30:58 scrubber: Error: unable to access VOB
neptune:/vobstore/src.vbs:
 ClearCase object not found
05/27/99 04:30:58 scrubber: Warning: skipping VOB
neptune:/vobstore/src.vbs
.
.
.
```

cmd-context **unregister –vob –uuid cb4caf2f.f48d11cc.abfc.00:01:53:00:e8:c3**

SEE ALSO

mktag, **mkview**, **mkvob**, **mount**, **register**, **registry_ccase**, **umount**

unreserve

Changes a reserved checkout to unreserved

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

unres·erve [**–vie·w** *view-storage-dir-pname*] [**–cact**]
[**–c·omment** *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**]
pname ...

DESCRIPTION

The **unreserve** command changes the checkout status of a checked-out version of an element to *unreserved*. A temporary `unreserve checkout of version` event record is written to the VOB database.

PERMISSIONS AND LOCKS

Permissions Checking: For each object processed, you must be one of the following: element group member, element owner, VOB owner, **root**. See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB, element type, element, branch type, branch.

OPTIONS AND ARGUMENTS

SPECIFYING THE VIEW. *Default:* The current view's checkout is changed (unless you specify an element with a view-extended pathname).

–vie·w *view-storage-dir-pname*
Specifies the view whose checkout is to be changed. For *view-storage-dir-pname*, use the view storage directory pathname listed by the **lscheckout –long** command. (The *host***:** prefix is optional.)

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more *event records*, with commenting controlled by your **.clearcase_profile** file (default: **–nc**). See *CUSTOMIZING COMMENT HANDLING* in the **comments** reference page. Comments can be edited with **chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* |**–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the option you specify. See the **comments** reference page.

SPECIFYING THE ELEMENTS. *Default:* None.

–cact
(UCM) Unreserves each checked-out version in the change set of the current activity in your view.

pname ...
One or more pathnames, each of which specifies an element. The checkout in the current view is changed, unless you use a view-extended pathname to specify another view.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Change the checkout status of an element to unreserved.

  cmd-context **unreserve util.c**

  ```
  Changed checkout to unreserved for "util.c" branch "/main".
  ```

- Change the checkout status of an element in another view to unreserved. Note that the view's storage area is on a remote host.

  cmd-context **lscheckout –long hello.c**

  ```
  10-Aug-98.16:59:25     Ellie Jackson (jackson.user@oxygen)
  checkout version “hello.c” from /main/37 (reserved)
  by view: jackson_fix (“oxygen:/usr/home/jackson/ccviews/fix.vws”)
  “merge from bugfix branch”
  ```

  cmd-context **unreserve –view oxygen:/usr/home/jackson/ccviews/fix.vws hello.c**

  ```
  Changed checkout to unreserved for "hello.c" branch "/main".
  ```

- Check out an element, check its status, and change its status to unreserved.

  cmd-context **co –nc edge.c**

  ```
  Checked out "edge.c" from version "/main/1".
  ```

  cmd-context **lscheckout edge.c**

  ```
  08-Dec.12:17 jackson checkout version "edge.c" from /main/1 (reserved)
  ```

  cmd-context **unreserve edge.c**

```
Changed checkout to unreserved for "edge.c" branch "/main".
```

SEE ALSO

checkin, **checkout**, **lscheckout**, **reserve**, **uncheckout**

update

Updates elements in a *snapshot view* or Attache workspace

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand
Attache	command

SYNOPSIS

- ClearCase and ClearCase LT only—Update elements using the graphical update tool:

  update –g·raphical [*pname ...*]

- ClearCase and ClearCase LT only—Update elements from the command line:

  update [**–print**] [**–f·orce**] [**–ove·rwrite** | **–nov·erwrite** | **–ren·ame**]
  [**–cti·me** | **–pti·me**] [**–log** *pname*] [*pname ...*]

- ClearCase and ClearCase LT only—Load elements from the command line by specifying one or more load rules:

  update –add·_loadrules [**–print**] [**–f·orce**] [**–ove·rwrite** | **–nov·erwrite** | **–ren·ame**]
  [**–cti·me** | **–pti·me**] [**–log** *pname*] *pname* [*pname ...*]

- Attache:

  update { [**–print** [**–since** *date_time*] |
  [**–all** |**–since** *date_time*]
  [**–ove·rwrite** | **–nov·erwrite**] [**–pti·me**] [**–compress**] }
  [**–r·ecurse**] [**–log** *pname*] *pname...*

DESCRIPTION

ClearCase and ClearCase LT—Updating Loaded Elements

For one or more loaded elements, the **update** command does the following:

- Reevaluates the *config spec* to select a versions of loaded elements in the VOB, and loads them if they differ from the currently loaded element versions
- Unloads the file or directory from the view if a loaded element is no longer visible (that is, a new directory version doesn't have an entry for the element). To unload a directory element, ClearCase and ClearCase LT

- Recursively delete all loaded elements
- Rename the directory to *directory-name*.**unloaded** if necessary, thus preserving all *view-private files* and *view-private directories*.

- If the version in the snapshot view is different from the version in the VOB selected by the config spec, copies the version selected by the config spec into the view. The version in the view can be different if, for example, the selected version in the VOB is newer, or if a label is attached to the selected version in the VOB, but not to the version in the view

update does not apply to files or directories that are checked out to the current view.

If **update** cannot access a VOB (perhaps due to problems in the network), any elements from that VOB remain loaded, but are put in a special state (`rule unavailable`).

The **update** command accounts for the fact that VOB elements specified by your config spec may change while an update is in progress. To avoid loading an inconsistent set of element versions, **update** ignores versions that meet both of the following criteria:

- The version is selected by a config spec rule that specifies the **LATEST** version label.
- The version was checked in after the moment the update operation began.

update also accounts for the fact that the system clocks on different hosts may not be synchronized.

When issued from a snapshot view,the following **cleartool** commands invoke **update** at the completion of the command:

- **edcs**
- **findmerge** (only when used to merge versions of a directory)
- **ln**
- **merge** (only when used to merge versions of a directory)
- **mkdir**
- **mkelem**
- **mv**
- **rmname**
- **setcs**
- **uncheckout**

ClearCase and ClearCase LT—Loading New Elements

The form of the **update** command that specifies the **–add_loadrules** option enables you to add new load rules to your **config_spec** and load the elements that those rules specify.

Attache

This command downloads the specified files to the workspace.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

ClearCase and ClearCase LT

USING THE GRAPHICAL UPDATE TOOL. *Default*: The update is performed in the command window.

–g·raphical
Invokes the graphical update tool.

USING THE PREVIEW MODE. *Default*: None.

–print
Produces a preview of the update operation: instead of copying or removing files, **update** prints a report to standard output of the actions it would take for each specified element.

CONFIRMATION STEP. *Default*: **update** prompts for confirmation of the elements to be updated. However, **update** does not in all circumstances prompt you to confirm all the elements to be updated. Sometimes there are no confirmation prompts when you update elements, even though you have not specified **–force**.

–f·orce
Suppresses the confirmation prompts.

HANDLING HIJACKED FILES. *Default:* Leaves all hijacked files in the view with their current modifications (**–noverwrite**).

–ove·rwrite
Overwrites all *hijacked files* with the version selected by the config spec.

–nov·erwrite
Leaves all hijacked files in the view with their current modifications.

–ren·ame·
Renames hijacked files to *filename***.keep** and copies the version in the VOB selected by the config spec into the view.

DETERMINING THE MODIFICATION TIMESTAMP. *Default:* The initial default is set by the **mkview** command. Thereafter, the most recently used time scheme is retained as part of the view's state and is used as the default behavior for the next update.

–cti·me
Sets the time stamp of a file element to the current time, that is, the time at which the version is copied into the view. **–ctime** has no effect on directories (directories always use the current time).

–pti·me
Sets the time stamp of a file element to the time at which the version was checked in to the VOB. –**ptime** has no effect on directories. (Directories always use the current time.)

SPECIFYING A FILE TRANSFER LOG. *Default:* **update** generates a log file and writes it to the root directory of the snapshot view.

–**log** *pname*
Specifies a log file for the operation. The log file lists the actions taken by the **update** command, as well as an indication of any errors that occur during the operation. Use **–log /dev/null** to suppress generation of the log file.

SPECIFYING NEW LOAD RULES. *Default:* None.

–add_loadrules
Specifies that the *pname* argument is a new load rule. The new rule is appended to the view's config spec, and the elements it specifies are loaded.

SPECIFYING THE ELEMENTS TO BE UPDATED OR ADDED. *Default:* If you do not specify **–add_loadrules**, the current snapshot view; if you specify **–add_loadrules**, none.

pname ...
If you do not specify **–add_loadrules**, this argument specifies the files and/or directories to update. All specified directories, including the root directory of the snapshot view, are updated recursively.

If you specify **–add_loadrules**, this argument is interpreted as a new load rule. The elements specified by the rule are loaded and the rule is appended to the config spec of the current view. *pname* must be either a pathname relative to your current location in the directory structure of the snapshot view or an absolute path that includes the snapshot view path.

Attache

SPECIFYING THE FILES TO BE UPDATED. *Default:* None.

pname...
Specifies the files, directories, and/or links to be updated. For a *pname* containing a symbolic link, Attache updates a copy of the file or directory the link points to, rather than the link itself. Wildcard patterns are expanded with reference to the view. In addition, arguments of the form @*pname* can be used to add the contents of the local file *pname* as pathname arguments. The pathname arguments can contain wildcards (most useful for excluding particular files; see the **wildcards** reference page), and must be listed in the file one per line, or also be of the form @*pname*. Specifying a relative pathname for @*pname* begins from Attache's start-up directory, not the working directory, so a full local pathname is recommended.

–all
Specifies that all files are to be downloaded to the Attache workspace.

–since *date_time*
Downloads to the Attache workspace all files checked in since the time specified in *date_time.*

DISPLAY FILES TO BE UPDATED. *Default:* None.

–**print** [**–since** *date_time*]
Displays the files that need updating, but does not update them in the Attache workspace. If –**print** is used, a reference time must be specified. **–since** displays files updated since *date_time*. A project config file which has been used to do an update can also be specified. The config file is specified as @*filename* for the *pname* argument. For each config file used to do an update, Attache remembers the last update time and uses it for the next update with that config file.

SPECIFYING HOW THE FILES ARE TO BE UPDATED. *Default:* When a directory is specified, its file contents are updated. If a destination file already exists that is identical in contents with the source file, it is not overwritten. If an existing destination file is read-only and differs from the source, it is always overwritten. If the destination file exists and is writable, an overwrite query is issued.

–ove·rwrite
Suppresses the query and causes all writable files to be overwritten.

–nov·erwrite
Suppresses the query and causes no writable file to be overwritten.

–pti·me
Causes the last-modified time stamp of the destination file to be set to that of the source file. –**ptime** has no effect on directories.

–compress
Causes files to be compressed while being uploaded and uncompressed after the transfer to improve performance over slow communications lines.

HANDLING OF DIRECTORY ARGUMENTS. *Default:* For each *pname* that specifies a directory element, **update** downloads to the Attache workspace the contents of that directory, but not the contents of any of its subdirectories.

–r·ecurse
Includes files from the entire subtree below any subdirectory included in the top-level listing. Directories are created as necessary and the current directory is taken into account if relative patterns are given.

SPECIFYING A FILE TRANSFER LOG. *Default:* None.

–**log** *pname*

Specifies a log file for the operation. The log file lists the workspace-relative pathname of each file transferred by the Attache **update** command, as well as an indication of any errors that occur during the operation. Log file pathnames are absolute, not relative to the current workspace root.

The log file can be used as an indirect file in a **get** command if there are errors which prevent the updating of all files.

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

ClearCase and ClearCase LT

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

- Update the file **./foo.c** using the current time as the time stamp.

  cmd-context **update –ctime foo.c**

- Update the current directory; if there are any hijacked files, rename them *filename***.keep** and copy the VOB versions specified by the config spec into the view.

  cmd-context **update –rename**

- Load into the current view the new elements in ./**vobs/doc/user_manual**, adding the rule `load /vobs/doc/user_manual` to the view's config spec.

  cmd-context **update –add_loadrules vobs/doc/user_manual**

Attache

- Determine which files have been changed since yesterday in the **/vobs/proj** VOB.

  cmd-context **update –print –since yesterday –r /vobs/proj**

- Update all files changed since yesterday in the **/vobs/proj** VOB, overwriting any writable files in the workspace.

  cmd-context **update –since yesterday –r –overwrite /vobs/proj**

SEE ALSO

checkin, **checkout**, **clearviewupdate**, **config_spec**, **edcs**, **get**, **findmerge**, **ln**, **merge**, **mkdir**, **mkelem**, **mv**, **rmname**, **setcs**, **uncheckout**

version_selector

Version-selector syntax

APPLICABILITY

Product	Command Type
ClearCase	general information
ClearCase LT	general information
Attache	general information

SYNOPSIS

branch-pathname/*version-number*
[*branch-pathname*]/ *label*
[*branch-pathname*/]{ *query* }

DESCRIPTION

A version selector identifies a version of an element in a version tree. You can use it with the **–version** command-line option, as part of a rule in a *config spec*, and as part of a *version-extended pathname*. The version selector has three general forms. Each identifies a version in a different way:

- By version-ID
- By the version label attached to it
- By a query on the meta-data attached to it, or some other version characteristic

A version selector selects one version of an element, no version of an element, or generates an error, if ambiguous.

Branch Pathnames

The *branch pathname* in a version selector identifies the branch on which a version resides. A branch pathname consists of a series of branch type names separated by slashes (/). The root of a version tree is the main branch (default name: **/main**), which must be the first entry in the branch pathname unless you use the ellipsis wildcard (not valid in version-extended pathnames). Examples:

`/main`	*(main branch)*
`/main/bugfix`	*(bugfix branch, off the main branch)*
`/main/motif/bugfix`	*(bugfix branch, off the /main/motif branch)*
`/main/motif/bugfix/jpb`	*(jpb branch, off the /main/motif/bugfix branch)*

SELECTION BY VERSION-ID

branch-pathname/version-number

Selects the version with the specified version-ID. This form requires a branch pathname.

Examples:

`/main/2`	*(version 2 on main branch)*
`/main/bugfix/5`	*(version 5 on bugfix branch off main branch)*
`/main/motif/bugfix/1`	*(version 1 on subbranch of /main/motif branch)*

In a version-extended pathname, the version-ID follows the element name and *extended naming symbol* (default: @@). For example:

`hello.c@@/main/4`	*(version 4 on main branch of file 'hello.c')*
`util.c@@/main/motif/experiment/1`	*(version 1 on the /motif/experiment branch, off the main branch)*
`include@@/main/4/hello.h/main/3`	*(version 3 on the main branch of file 'hello.h', in version 4 on the main branch of directory 'include')*

RESTRICTION: In a version-extended pathname, you cannot use the ellipsis wildcard (**...**):

`include.h@@/.../bugfix/REL2`	*(is not valid)*

SELECTION BY VERSION LABEL

[*branch-pathname*]/*label*

Selects the version with the specified *label*. The branch pathname is optional, but the slash is required. Examples:

`/main/LATEST`	*(most recent version on main branch)*
`.../bugfix/REL2`	*(version labeled REL2 on a branch named bugfix, at any branching level)*
`/main/bugfix/REL2`	*(version labeled REL2 on a bugfix branch that is a subbranch of main)*
`/main/sunport/openlook/BUG3`	*(version labeled BUG3 on a particular third-level branch)*
`REL2`	*(version labeled REL2 on any branch)*

RESTRICTION: In a version-extended pathname, you cannot use the ellipsis wildcard (**...**):

`include.h@@/.../bugfix/REL2`	*(is not valid)*

The label **LATEST** is predefined; it evaluates to the most recent version on each branch of an element. If the most recent version on the main branch is version 4, these two version selectors identify the same version:

```
/main/LATEST
/main/4
```

A version selector can consist of a standalone label, such as **REL2**. Standalone labels can be ambiguous, however. For example, **/main/bugfix/REL2** and **REL2** may or may not be equivalent for a given element:

- If the **REL2** label type was created as one-per-element (default), the two version selectors must be equivalent.
- If **REL2** was created with **mklbtype –pbranch**, however, the label can be used once per branch. If the label is actually attached to two or more versions of an element, an error occurs. No error occurs for elements that happen to have only one instance of a one-per-branch label type.

Version Labels As Hard Links

Version labels appear as UNIX hard links in an element's directory tree in *version-extended namespace*. (See the **pathnames_ccase** reference page.) If a version label was defined to be one-per-element, an additional hard link appears at the top level of an element's directory tree. For example, if **BL3** is a one-per-element label, these version-extended pathnames are both unambiguous references to the same version:

```
hello.c@@/BL3
hello.c@@/main/bugfix/patch2/BL3
```

In effect, this feature allows you to reference a version without knowing its exact location in the version tree.

If a label was defined with the **–pbranch** option, it does not appear in the element's top-level extended namespace directory (as implied earlier). Thus, if the one-per-element label, **BL3**, and the one-per-branch label, **TEST_LBT**, was attached to version **/main/1** of file **hello.c**, its top-level extended namespace directory would look like this:

```
% cd hello.c@@

% ls

BL3 main
```

SELECTION BY QUERY

[*branch-pathname*/]{*query*}

Selects the version that satisfies the specified query. The branch pathname is optional.

The query expression consists of one or more query primitives and operators, organized according to the syntax rules listed in the **query_language** reference page. Enclose the query expression in braces ({ }).

Enclose the entire version selector in single quotes (' ')—or double quotes (" ") if it includes spaces or characters that have special meaning to the shell. Use double quotes to set off string literals within the query expression.

`/main/{TESTED=="yes"}`	*(the latest version on the main branch for which the 'TESTED' attribute has the value 'yes')*
`{hltype"(design_spec,<-)"}`	*(the version on any branch that is the 'to' end of a hyperlink of type 'design_spec')*
`/main/bugfix/"{!lbtype(REL2)}"`	*(on bugfix branch, the latest version that is not labeled 'REL2)*
`"{created_by(jpb)&&pool(sr1)}"`	*(the version on any branch created by user 'jpb' which is stored in the 'sr1' storage pool)*

If the version selector includes a branch pathname, the **view_server** selects the latest version on the branch that satisfies the query. If the version selector does not include a branch pathname, the **view_server** selects the version on any branch that satisfies the query. However, without a branch pathname, a query is ambiguous when more than one version of the element satisfies the query; versions on different branches, or two versions on the same branch, for example.

The version-selection operation fails if the query selects no version or is ambiguous.

A version-extended pathname can include a query, but is subject to the same restrictions as other version selectors of this form. That is, the query must select exactly one version to succeed. For example, this command displays the most recent version that has an attribute of type **TESTED**:

```
% cat include.h@@/"{attype(TESTED)}"
```

Note the use of quotes to prevent the shell from interpreting the brace and parenthesis characters. As an alternative, you can quote the entire pathname:

```
% cat "include.h@@/{attype(TESTED)}"
```

If multiple branches have versions with a **TESTED** attribute, the version selector used in the examples above is ambiguous, and an error occurs.

RESTRICTION: In a version-extended pathname, you cannot use both a branch pathname and a query:

```
% cat "include.h@@/main/{attype(TESTED)}"
```
(is not valid)

```
% cat "include.h@@/main/rel2_bugfix/{attype(TESTED)}"
```
(is not valid)

You can use the **describe** command to work around this restriction:

```
% cat `cleartool describe -s -ver /main/rel2_bugfix/"{attype(TESTED)}" include.h`
```

SEE ALSO

config_spec, **pathnames_ccase**, **query_language**

view

Data structure for views

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure
Attache	data structure

SYNOPSIS

UNIX directory tree created by **mkview** command

DESCRIPTION

A *view* provides a workspace in which users can access versions of elements and other file-system objects that are outside of ClearCase or ClearCase LT source control. There are two kinds of views:

- *Dynamic views*, which provide transparent access to versions of *elements* in the *VOB* and to *view-private objects*. Each time you access an element through a dynamic view, the view's **view_server** process evaluates the view's *config spec* and selects a particular version of the element. Thus, such a view updates itself with new versions created in other views. Only ClearCase and Attache support dynamic views.
- *Snapshot views*, which contain copies of versions of specified elements, along with view-private objects. The view never updates itself with new versions created from other views. Instead, the **update** command reevaluates the view's config spec and loads the newly selected versions into the view. ClearCase LT supports only snapshot views.

This reference page discusses both a view's physical data structures and the way file system data appears to a user process through a view. Each type of view is discussed separately.

Dynamic Views—The File System

A dynamic view is an *MVFS* (multiversion file system) directory tree that enables dynamic access to VOB elements.

Dynamic Views—View Storage Directory

A dynamic view is implemented as a standard directory tree, whose top-level directory is termed the view storage directory. The directory contains these files and subdirectories:

.access_info	A file of view access event information that is periodically updated by the view server.
.pid	A one-line text file that lists the process-ID of the associated **view_server** process, currently running on the host where the view storage directory resides.
.view	A file that lists the view's universal unique identifier (UUID) and attribute For descriptions of the other lines in this file, see *VIEW CONFIGURATION* in the **view_server** reference page.
admin	A directory that contains administrative data related to the amount of disk space a view is using. Use **space –view** to list this data.
config_spec	A file that stores the view's current config spec, in the form displayed by **catcs**.
.**compiled_spec**	A modified version of **config_spec**, which includes accounting information.
.**identity**	A subdirectory whose files establish the view's owner and group memberships. Only the view's creator has access rights to this subdirectory. It has the same structure as the like-named subdirectory within a VOB storage directory. See the **vob** reference page for more information.
.s	A subdirectory that implements the view's private storage area. See the *Dynamic Views—Private Storage Area* section.
db	A subdirectory containing the files that implement the view's embedded database. See *Dynamic Views—View Database*.

Dynamic Views—Private Storage Area

Subdirectory **.s** of the view storage directory is the root of a subtree that implements the view's private storage area. If the view was created with **mkview –ln**, then **.s** is actually a (UNIX-level) symbolic link, pointing to a remote storage area. The private storage area holds several kinds of objects:

VIEW-PRIVATE OBJECTS. A view-private object is a file-system object—file, directory, or link—created by a standard program within a VOB directory. Such objects are stored only within the view's private storage area. No VOB maintains any record of such objects.

NOTE: You cannot make view-private files that represent types of file system objects that are not mentioned above. For example, you cannot create the following types of view-private files: named pipes, character device files, or block device files.

CHECKED-OUT FILES. A checked-out version is a file created by the **checkout** command. This file is an editable copy of the version being checked out.

A checked-out version is very much like a view-private file, except that there is a corresponding object in the VOB database: the special "placeholder" version with the **CHECKEDOUT** version label.

UNSHARED DERIVED OBJECTS. An unshared derived object is a data container created by execution of a **makefile** build script by **clearmake** or by any program invoked by **clearaudit**. A corresponding *derived object* is created in the VOB database if you are using a dynamic view.

NOTE: An unshared derived object remains in the view's private storage area even after the DO becomes shared. *Promotion* of the DO involves copying, not moving, the data container. The **winkin** command and **view_scrubber** utility remove unshared derived objects from a view's private storage area.

NONSHAREABLE DERIVED OBJECTS. A nonshareable derived object is a data container created by execution of a **makefile** build script by **clearmake** or by any program invoked by **clearaudit**, from a dynamic view. No information about the DO is stored in the VOB database. When you use **winkin** or **view_scrubber –p** to convert a nonshareable DO to a shareable DO, the command promotes the DO's data container to the VOB, and removes the data container from view storage.

CONFIGURATION RECORDS. The file **view_db.crs_file** in the **.s** subdirectory is actually part of the view's database, as described in the following section. It is the part that stores the configuration records of derived objects built in the view.

STRANDED FILES. The directory **lost+found** in the **.s** subdirectory contains stranded files, that is, files that were view-private and are no longer accessible through normal ClearCase or Attache operations. (Only dynamic views have **lost+found** directories.) A file becomes stranded when there is no VOB pathname through which it can be accessed. For example:

- A VOB can become temporarily unavailable, for example, by being unmounted.
- A VOB can become permanently unavailable, for example, by being deleted.
- A VOB directory can become permanently unavailable, by being deleted with a **rmelem** command.

See the description of the **recoverview** command for more information about recovering stranded files (**recoverview** does not apply to snapshot views.)

Dynamic Views—View Database

The view database subdirectory, **db**, contains these files:

view_db.dbd	A compiled database schema, used by embedded DBMS routines for database access. The schema describes the structure of the view database. The **mkview** command creates this file by copying *ccase-home-dir***/etc/view_db.dbd**.

view_db_schema_version	A schema version file, used by embedded DBMS routines to verify that the compiled schema file is at the expected revision level. The **mkview** command creates this file by copying *ccase-home-dir*/**etc/view_db_schema_version**.
view_db.d0*n* **view_db.k01**	Files in which the database's contents are stored.
vista.*	Database control files and transaction logs.
view_db.crs_file	Stores the configuration records of nonshareable and unshared derived objects. This file resides in subdirectory **.s** of the view storage directory, allowing it to be remote. Compressed copies of the configuration records are cached in a view-private file, **.cmake.state**, located in the directory that was current when the build started. This speeds up configuration lookup during subsequent builds in the view.

The view database keeps track of the objects in its private storage area: view-private objects (files, directories, and links), checked-out versions, nonshareable derived objects, and unshared derived objects.

Snapshot Views—The File System

The snapshot view directory tree is part of the native file system (as opposed to the directory tree of a dynamic view, which is part of the *MVFS*, or multiversion file system). In addition to copies of ClearCase and ClearCase LT elements, the root of this directory tree (referred to as the snapshot view's root directory) contains the following files and subdirectories:

.view.dat	A read-only text file used to identify the current directory as part of a view.
.view.stg or a generated directory name	A directory used to maintain the view. For a colocated view storage directory only, the default name of this directory is named **.view.stg**; for a noncolocated view storage directory, this directory has a generated name. Certain view configurations require that this directory be located somewhere outside the snapshot view's root directory. (Refer to the **mkview** reference entry for information on view configurations.)

NOTE: When referring to a snapshot view, many ClearCase and ClearCase LT commands require the *snapshot-view-directory-pname* argument (rather than an argument specifying the view-tag). By reading the view's **.view.dat** file, which is in the root directory of the snapshot view, ClearCase and ClearCase LT can find the view storage directory.

Snapshot Views—View-Storage Directory

The snapshot view's view-storage directory is used by ClearCase and ClearCase LT to maintain the view. It contains the following files and subdirectories:

.access_info	A file of view access event information that is periodically updated by the view server.
admin	A directory that contains administrative data related to the amount of disk space a view is using. Use **space –view** to list this data.
.pid	A one-line text file that lists the process-ID of the associated **view_server** process, currently running on the host where the view storage directory resides.
.view	A file that lists the view's universal unique identifier (UUID) and attributes.
config_spec	A file that stores the view's current config spec, in the form displayed by **catcs**.
.**compiled_spec**	A modified version of **config_spec**, which includes accounting information.
.identity	A subdirectory whose files establish the view's owner and group memberships. Only the view's creator has access rights to this subdirectory. It has the same structure as the like-named subdirectory within a VOB storage directory. See the **vob** reference page for more information.
.s	A subdirectory containing empty subdirectories.
db	A subdirectory containing the files that implement the view's embedded database. See *Snapshot Views—View Database*.
view_db.state	A file that records the current state of the view.

Snapshot Views—View Database

The view database subdirectory, **db**, contains these files:

view_db.dbd	A compiled database schema, used by embedded DBMS routines for database access. The schema describes the structure of the view database. The **mkview** command creates this file by copying *.ccase-home-dir***/bin/view_db.dbd**.
view_db_schema_version	A schema version file, used by embedded DBMS routines to verify that the compiled schema file is at the expected revision level. The **mkview** command creates this file by copying *ccase-home-dir***/bin/view_db_schema_version**.
view_db.d0*n* **view_db.k01**	Files in which the database's contents are stored.
vista.*	Database control files and transaction logs.

For a snapshot view, the view database keeps track of the loaded VOB objects and checked-out versions in the view.

ClearCase and Attache Only—Comparison of Dynamic and Snapshot Views

The information in this section is intended for ClearCase and Attache users, who can choose between dynamic and snapshot views.

Snapshot views:

- Require an update operation periodically
- Provide optimal performance for developing source files and for prototype builds
- Facilitate working from a remote location
- Use the **update** command to copy a specified set of source files into the view

Work in a snapshot view when any of these conditions is true:

- You want to optimize build performance to achieve native build speed.
- You want to work with source files under ClearCase control when you are either disconnected from the network or connected to the network intermittently from a remote location.
- You want to access a view from a computer that is not a ClearCase host.
- Your development task doesn't require the ClearCase *build auditing* and *build avoidance* features.

Dynamic views:

- Show changes to elements in the VOB as they are made (and thus do not require an update operation)
- Support the ClearCase build auditing and build avoidance features
- Provide access to all elements in all mounted VOBs without using **update**

Work in a dynamic view when any of these conditions is true:

- Your development task requires audited builds.
- You want to enable derived object sharing.

  You can use express builds to enable DO reuse in your view, but not allow the DOs to be winked in by other views. For more information, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*.
- You prefer to use a view that is updated during development.

Table 15 compares features of snapshot views and dynamic views.

Table 15 A Comparison of View Features

Feature	For dynamic views	For snapshot views
Storage requirements	You need enough disk space to store any view-private files or view-private directories.	You need enough disk space to store the copies of the loaded elements and any view-private files or view-private directories.
To access ClearCase elements	You start the view and mount VOBs on your computer.	Your view's config spec loads elements into the view.
Version selection	For any element in a mounted VOB, the view always selects the version specified by the config spec. You can traverse the *view-extended namespace* to see any version of any element in any mounted VOB.	You use the **update** command to update the elements loaded into the view. You can see only the version of an element that is loaded into your view.
VOB symbolic links and VOB hard links	The link is traversed each time you access the object.	The version to which the link resolves at update time is copied into the view.
Version that is checked out	By default, the checkout operation checks out the latest version on a branch, regardless of what the config spec specifies.	The checkout operation checks out the file loaded in your view, which may not be the latest version on a *branch*.
Build features	You can use the **clearmake** features of build auditing and build avoidance.	You can use **clearmake**, but the build auditing and build avoidance features are disabled.
Sharing the views with other team members	View objects are accessible to other team members through ClearCase's view-extended namespace.	View objects are accessible to other team members if you tell themthe snapshot view directory path. Snapshot views are not accessible through ClearCase's view-extended namespace.

SEE ALSO

config_spec, **endview**, **lsview**, **mkview**, **mvfsstorage**, **recoverview**, **reformatview**, **registry_ccase**, **scrubber**, **setview**, **startview**, **update**, **view_scrubber**, **vob**

view_scrubber

Remove derived object data containers from dynamic view storage

APPLICABILITY

Product	Command Type
ClearCase	command

SYNOPSIS

ccase-home-dir/**etc/view_scrubber** [**–p**] [**–k**] [**–n**] [*DO-pname ...*]

DESCRIPTION

The **view_scrubber** program cleans a view's private storage area by removing data containers for derived objects (DOs).

NOTE: This command does not apply to snapshot views because derived objects can be created only in dynamic views.

WARNING: This command modifies the way in which view-resident objects are combined with VOB-resident objects to produce a virtual workspace. To avoid errors, make sure that no application or development tool is using the view's files when this command is executed.

The most common way to run the **view_scrubber** is indirectly, by running the *ccase-home-dir*/**etc/view_scrubber.sh** script supplied with ClearCase.

Scrubbing is useful in the situations described in the following sections.

Cleaning Up after a Winkin

When a **clearmake** build winks in a shareable DO for the first time, the DO's data container is copied from the private storage area of the view in which it was built to the VOB storage pool. At this point:

- The view where the DO was originally built continues to use the data container in view storage.
- Any other view to which the DO is subsequently winked in uses the data container in VOB storage.

Running **view_scrubber** in the view where the DO was built simplifies the situation. **view_scrubber** performs the following steps:

1. Removes the DO with **rm(1)**. This deletes the data container from view storage.
2. Winks in the DO to the view, which establishes a link to the data container in VOB storage.

Now, all views that share the DO access the data container in VOB storage. This eliminates the redundant, space-consuming data container in view storage.

Self-Winkin

By default, the data container for a nonshareable DO or an unshared DO remains in view storage until the DO is deleted or overwritten. **view_scrubber –p** transfers the data container to VOB storage, thus freeing space in the view storage area. In essence, this involves winking in the DO to the same view. **view_scrubber –p** performs the following steps:

1. (Nonshareable DO only) Converts the DO to a shareable DO by writing information about the DO into the VOB.

   If the DO has any sub-DOs or siblings, **view_scrubber –p** makes them shareable.
2. Promotes the data container from view storage to VOB storage.
3. Removes the DO with **rm(1)**, which deletes the data container from view storage.
4. Winks in the DO to the view, which establishes a link to the data container in VOB storage.

You can also use the **winkin** command to accomplish this scenario.

NOTE: When a nonshareable DO is converted to a shareable DO, its DO-ID changes. For more information, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*.

OPTIONS AND ARGUMENTS

PREPROCESSING WITH A PROMOTION. *Default:* **view_scrubber** removes view-resident data containers, then restores the derived objects to the view through winkin. *Requirement:* The derived objects' data containers must already be in VOB storage.

–p
: Before performing the default processing described above, promotes (copies) the derived objects' data containers from view storage to VOB storage. This removes the requirement that the data containers be in VOB storage.

ERROR RECOVERY. *Default:* **view_scrubber** aborts if it is unable to complete its work on any derived object.

–k
: Keeps going, even if one or more derived objects cannot be processed successfully.

NO-EXECUTE OPTION. *Default:* **view_scrubber** performs its work and displays appropriate messages.

–n
: Suppresses the actual processing of data containers. **view_scrubber** displays messages describing the work it would have performed.

DERIVED OBJECTS TO PROCESS. *Default:* If you don't specify any DOs as command arguments, **view_scrubber** reads a one-per-line list of pathnames from **stdin**, which must be a pipe.

DO-pname ...
One or more standard pathnames of derived objects.

EXAMPLES

- Make the view to be scrubbed the current working view, and move to the directory of interest. Then scrub DO containers for the entire directory tree, using the script *ccase-home-dir***/etc/view_scrubber.sh** (which invokes the **view_scrubber** program).

  % **cleartool setview big_view**

  % **cd /vobs/src**

  % *ccase-home-dir***/etc/view_scrubber.sh**

- Scrub two DOs, promoting the data containers to VOB storage.

  % *ccase-home-dir***/etc/view_scrubber –p /view/cep/vobs/dev/lib/cmd.h **
  /view/smg/vobs/dev/lib/cmd_api.h

SEE ALSO

clearmake, **promote_server**, **scrubber**, **winkin**

view_server

Server process that performs version selection for a view

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

DESCRIPTION

A **view_server** is a long-lived process that manages activity in a particular *view*. It interprets the rules in the view's *config spec*, and (for *dynamic views*) tracks modifications to view-private files for other ClearCase and ClearCase LT software.

Each view requires a dedicated **view_server** on the host where the view storage area resides. The **view_server** is started by its host's **albd_server** process when necessary. It runs with the user-ID of the owner of the view storage directory (usually, the user who created the view). A **view_server** remains active until it is terminated by an **endview –server** command, a system shutdown, or a **kill(1)** command.

For a dynamic view, a **view_server** handles MVFS file-system requests (such as create, delete, and rename) by querying one or more VOB databases and comparing them against the view's own database. Using the view's config spec, it selects versions of file elements and directory elements to be in the view. It also handles requests from **cleartool**, **clearmake**, and **clearaudit** to look up VOB-database objects and/or names.

The **view_server**'s procedure for resolving element names to versions is as follows:

1. User-level software (for example, an invocation of the C compiler) references a pathname. The ClearCase MVFS, which processes all pathnames within VOBs, passes the pathname to the appropriate **view_server** process. (The *multiversion file system* on a ClearCase client host is linked (statically or dynamically) with the operating system kernel.)
2. The **view_server** attempts to locate a version of the element that matches the first rule in the config spec. If this fails, it proceeds to the next rule and, if necessary, to succeeding rules until it locates a matching version.
3. When it finds a matching version, the **view_server** selects it and has the MVFS pass a handle to that version back to the user-level software.

The **view_server** and the MVFS use caching techniques to reduce the computational requirements of this scheme. See *VIEW CACHES* on page 1025.

For a dynamic view, a **view_server** manages its view's database by tracking changes to objects in the view against the related objects in VOB databases (for example, the file in the view that corresponds to the checked-out version of a file element).

VIEW CONFIGURATION

When it begins execution, a **view_server** reads configuration information from the **.view** file in the view-storage directory. This is a text file that contains this information:

- Line 1: the location of the view storage directory, in *hostname*:*pathname* format
- Line 2: the view's UUID (unique identifier), which must not be changed
- Line 3: the *hostname* specified in line 1

NOTE: Lines 1 and 3 are placed in the **.view** file when the view is created, but the **view_server** ignores these lines thereafter.

The configuration file can include additional entries, each on a separate line:

–nshareable_dos
Specifies that builds in the view create nonshareable DOs. You can modify this property with the **chview** command.

–cache *size-in-bytes*
Sets the total size of the **view_server** caches to be *size-in-bytes*. The default is 500 KB for 32-bit platforms and 1 MB for 64-bit platforms. See *VIEW CACHES* on page 1025.

–textmode { **insert_cr** | **transparent** | **strip_cr** }
Specifies the interop text mode of the view. See the **mkview** reference page for more information.

–readonly
Applicable to dynamic views only. Prevents modification of the view's private data-storage area. A read-only view cannot be used for checkouts or for builds, because these operations create new files in view-private storage. (A **checkout** command will succeed in creating a checked-out version in the VOB database, but cannot create the corresponding view-private file.) You can modify this property with the **chview** command.

Snapshot views do not implement the private data-storage area, so **-readonly** does not prevent the view from being used for checkouts.

NOTE: –readonly does not prevent users from changing a view's config spec. Use view access permissions to implement this kind of restriction. (See the **mkview** reference page for details.)

–snapshot
Applicable to snapshot views only. Specifies that the view is a snapshot view.

–ptime

Applicable to snapshot views only. Specifies that time stamps on files copied into the view should show the time at which the version was created (rather than the current time).

VIEW CACHES

The **view_server** maintains a number of caches, consisting mostly of data retrieved from the VOB, to respond faster to RPCs from client machines. These are the view caches:

- Object cache, which facilitates retrieval of VOB and view objects (for example, versions and branches)
- Directory cache, which facilitates file system directory (readdir) access
- Stat cache, which stores file attributes
- Name cache, which stores names (of existing files and names that do not exist in a directory) and accelerates name lookups

When a **view_server** process is started, it chooses its cache size from the first one of these that yields a value:

- **–cache** directive in **.view** file (set with **mkview –cachesize** or **setcache –view –cachesize**)
- Contents of the file **/var/adm/atria/config/view_cache_size** (decimal number; set with **setcache –view –host**)
- Site-wide cache default stored in the registry (set with **setcache –view –site**)
- Default value: 500 KB on 32-bit platforms, 1 MB on 64-bit platforms

The cache size is allocated among the individual caches.

See the **setcache** reference page for information about setting the cache size for a view and for a site, and the **getcache** reference page for information about displaying cache information. For more information on optimizing performance, see the chapters on performance tuning in *Administering ClearCase*.

VIEW PERFORMANCE STATISTICS

The following command causes a **view_server** to write cumulative cache-performance (and other) statistics to its log file, **/var/adm/atria/log/view_log**:

kill -**HUP** *view_server-process-ID*

It then resets all the statistics accumulators to zero. You must enter this command on the host where the **view_server** executes.

SEE ALSO

albd_server, **endview**, **mkview**, **setview**, **setcache**, **startview**, **view**, **kill(1)**

VOB

Versioned object base data structures

APPLICABILITY

Product	Command Type
ClearCase	data structure
ClearCase LT	data structure
Attache	data structure

SYNOPSIS

UNIX directory tree created by **mkvob** command

DESCRIPTION

A *VOB* (*versioned object base*) is a data repository for a directory tree. Users access a VOB through a dynamic or snapshot view (see the **view** reference page). This reference page discusses both a VOB's physical data structures and its logical structures, as seen by a user process through a view.

A VOB is implemented as a standard directory tree, whose top-level directory is the VOB storage directory. The directory contains files and subdirectories:

.pid A one-line text file that lists the process-ID of the associated **vob_server** process, currently running on the host where the VOB storage directory resides.

admin A directory that contains administrative data related to the amount of disk space a VOB and its *derived object*s are using. Use **space –vob** to list this data.

vob_oid A one-line text file that lists the VOB's universal unique identifier(UUID). (OID means "object identifier".) This UUID is the same for all the replicas in a *VOB family* (ClearCase MultiSite). This file is also stored in the VOB's database; do not edit it.

replica_uuid A one-line text file that lists the replica UUID of this particular replica of the VOB. Different replicas created with ClearCase MultiSite have different identifiers.

.identity A subdirectory whose files establish the VOB's owner and group memberships. See *The .identity Directory* on page 1027.

s A subdirectory in which all of the VOB's local source storage pools reside.

d	A subdirectory in which all of the VOB's local derived object storage pools reside (dynamic views only).
c	A subdirectory in which all of the VOB's local cleartext storage pools reside. See *VOB Storage Pools* below.
db	A subdirectory containing the files that implement the VOB's embedded database. See *VOB Database* on page 1028.
vob_server.conf	A text file that records configuration information for the **vob_server**; read at **vob_server** startup. This file contains the setting for deferred deletion of source containers; deferred deletion is activated if you have turned on semi-live backup. See the **vob_snapshot_setup** reference page for more information.

The sections that follow discuss the more significant files and subdirectories in a VOB.

The .identity Directory

Subdirectory **.identity** records the VOB's ownership and group membership information. This directory has a very restricted access mode: only the user who is the VOB owner has any access rights. (As always, the host's **root** user can also access this directory.)

The **.identity** directory contains these files:

uid	The owner of this file is the VOB owner.
gid	The group to which this file belongs is the VOB's principal group.
group.*nn*	Each additional file (if any) indicates by its group membership an additional group on the VOB's group list. In addition, the file's name identifies the group by numeric ID (**group.30**, **group.2**, and so on).

You can use **describe** to display this information.

Example: A VOB is created by user **drp**, whose password entry places him in the **vga** group, and who also belongs to groups 2 (**bin**) and 30 (**dvt**). The VOB storage directory's **.identity** subdirectory contains these files to record this information:

```
-r----S--- 1 drp vga 0 Oct 9 09:34 gid
-r----S--- 1 drp bin 0 Oct 9 09:34 group.2
-r----S--- 1 drp dvt 0 Oct 9 09:34 group.30
-r-S------ 1 drp vga 0 Oct 9 09:34 uid
```

root subsequently adds group 50 (**rlsgrp**) to the VOB's group list:

```
% cleartool protectvob -add_group rlsgrp /vobstore/...
```

This change is recorded by an additional file in **.identity**:

```
-r----S--- 1 drp rlsgrp 0 Oct 9 09:34 group.50
```

VOB Storage Pools

Each VOB storage directory is created with default storage pools, located within the directories listed above.

s/sdft	Default source storage pool, for permanent storage of versions' file system data.
c/cdft	Default cleartext storage pool, for temporary storage of the **cleartext** versions currently in use (for example, reconstructed versions of **text_file** elements).
d/ddft	Default derived object storage pool, for storage of promoted/shared derived objects (dynamic views only).

For more information on storage pools, see the **mkvob**, **mkpool**, and **chpool** reference pages.

VOB Database

The VOB database subdirectory, **db**, contains these files:

vob_db.dbd	A compiled database schema, used by embedded DBMS routines for database access. The schema describes the structure of the VOB database. The **mkvob** command creates this file by copying *ccase-home-dir***/etc/vob_db.dbd**.
vob_db_schema_version	A schema version file, used by embedded DBMS routines to verify that the compiled schema file is at the expected revision level. The **mkvob** command creates this file by copying *ccase-home-dir***/etccvob_db_schema_version**.
vob_db.d0*n* **vob_db.k0***n*	Files in which the database's contents are stored.
vista.*	Database control files and transaction logs.
db_dumper	Backup copy of *ccase-home-dir***/etc/db_dumper**. **reformatvob** invokes this copy of **db_dumper** only if it cannot invoke **/usr/atria/etc/dumpers/db_dumper.***num*, where *num* is the revision level of the VOB. See the **db_dumper** reference page for more information.
vob_db.str_file	Database string file that stores long strings.

CAUTION: Do not move the VOB database directory (**db**) to another host. Two server processes, **db_server** and **vobrpc_server** access a VOB's database. These two server processes and the VOB database directory must be on the host where the VOB storage directory physically resides. Moving the VOB database directory can negatively affect VOB performance and the integrity of the VOB database by recovering from interrupted transactions.

BACKUP DATABASE SUBDIRECTORIES. reformatvob does its work by creating a new VOB database. By default, it preserves the old database by moving it aside to a date-stamped name. Thus, a VOB storage directory may contain old (and usually unneeded) VOB database subdirectories, with

names like **db.0318**. If **reformatvob** is interrupted, it may leave a partially reformatted database with the name **db.reformat**.

LOGICAL DATA STRUCTURES

From the user's standpoint, a VOB contains file system objects and metadata. Some metadata is stored in the form of objects; other metadata is stored as records or annotations attached to objects.

The VOB Object and Replica Objects

Each VOB database contains a VOB object that represents the VOB itself. The VOB object provides a handle for certain operations. For example:

- Listing event records of operations that affect the entire VOB (see the **lshistory** command). This includes creation and deletion of type objects, removal of elements, and so on.
- Placing a lock on the entire VOB (see the **lock** command).

Using ClearCase MultiSite, you can create any number of replicas of a VOB at different sites. Each VOB replica is represented in the VOB database by a replica object.

File System Objects

A VOB database keeps track of users' file system objects using the following database objects:

File element	An object with a version tree, consisting of *branches* and *versions*. Each version of a file element has file system data: a sequence of bytes. Certain element types constrain the nature of the versions' file system data; for example, versions of **text_file** elements must contain text lines, not binary data.
Directory element	An object with a version tree, consisting of *branches* and *versions*. Each version of a directory element catalogs a set of file elements, directory elements (subdirectories), and VOB symbolic links. An extra name for an element that is already entered in some other directory version is termed a VOB hard link.
VOB symbolic link	An object that contains a text string. This string is interpreted by standard commands in the same way as an operating system symbolic link.

LINK COUNTS FOR FILE SYSTEM OBJECTS. Link counts for file system objects are stored in the VOB database, and are reported by the standard **ls(1)** command, as follows:

Symbolic link	1
File element	1
File version	1
Directory element	2

Directory version	2 plus number of directory elements cataloged in that version
Branch	2 plus number of subbranches (each branch appears as a subdirectory in the extended-namespace representation of an element's version tree)

This scheme satisfies two UNIX rules:

- The link count is at least one for an object that has a name.
- The link count of a directory is 2 + *number-of-subdirectories*.

The scheme does not satisfy the rule that the link count should be the number of names the object has in the current namespace.

Type Objects

A VOB can store several kinds of type objects:

Element type	Defines a class of elements within the VOB.
Branch type	Defines a set of like-named branches in some or all of the VOB's elements.
Label type	Defines a mnemonic name that can be attached to a set of versions, thus defining a configuration of the VOB's elements.
Attribute type	Defines a name to be used in attaching name/value pairs to VOB-database objects.
Hyperlink type	Defines a class of logical arrow that can be used to connect pairs of objects.
Trigger type	Defines a monitor on operations that modify the VOB's objects.

Instances of Type Objects

After a type object is created, users can create any number of instances of the type.

Element	Each file or directory element in a VOB is created by **mkelem** or **mkdir** as an instance of an existing element type in that VOB.
Branch	Each branch in an element is created by **mkbranch** as an instance of an existing branch type in that element's VOB.
Version label	The **mklabel** command annotates a version with a version label, by creating an instance of an existing label type.
Attribute	The **mkattr** command annotates a version, branch, element, VOB symbolic link, or hyperlink with an attribute, by creating an instance of an existing attribute type. Each instance of an attribute has a particular value—a string, an integer, and so on.
Hyperlink	The **mkhlink** command creates a hyperlink object, which is an instance of an existing hyperlink type. A typical hyperlink connects two objects, in the same VOB or in different VOBs.

Trigger	The **mktrigger** command creates a *trigger* object, which is an instance of an existing trigger type. The trigger becomes attached to one or more elements.

ClearCase Dynamic Views Only—Derived Objects

A VOB's database stores information on all the shareable derived objects (DOs) created at pathnames within the VOB. For each DO, the database catalogs this information:

- The directory element, along with the location of the DO within the directory (for example, **util.o** or **sun5/util.o**)
- The DO's unique identifier, its DO-ID
- Shopping information for the DO

DOs are accessible only through a dynamic view. For more information, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*.

ClearCase Dynamic Views Only—Configuration Records

A VOB's database stores the configuration records (CRs) associated with shareable derived objects and DO versions (derived objects that have been checked in as versions of elements). Each CR documents a single target-rebuild, which typically involves execution of one build script. For more information, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*.

Event Records

Nearly every operation that modifies the VOB creates an event record in the VOB database. See the **events_ccase** reference page for more information.

The **vob_scrubber** utility deletes unneeded event records. By default, the scheduler runs **vob_scrubber** periodically. See the **schedule** reference page for information on describing and changing scheduled jobs.

The VOB Root and lost+found Directories

When it is first created by the **mkvob** command, a VOB appears to users as an almost-empty directory tree. It contains no files, and only two directories: the VOB root (ClearCase, ClearCase LT, and Attache) and **lost+found** (ClearCase and Attache only).

VOB Root Directory — **mkvob** executes a **mkdir** command to create a directory element, the VOB root directory, in the new VOB. Mounting a VOB makes its root directory accessible at the VOB-tag (VOB mount point) pathname.

For most purposes, the VOB root directory is like any other directory element you subsequently create within the VOB. But there are differences in certain contexts:

- The file-name pattern in a config spec rule cannot be a relative pathname that begins at a VOB root directory. A relative pathname must start below a VOB root directory. See the **config_spec** reference page for details.
- You must use a special syntax for a version-extended name that specifies a location in the version tree of a VOB's root directory:

  ls /usr/src/proj@@/main/3 *(invalid if directory 'proj' is VOB root)*

  ls /usr/src/proj/.@@/main/3 *(valid)*

The VOB root directory is assigned to the three default storage pools. All newly created file and directory elements are assigned to the default storage pools until new pools are created and assigned.

The lost+found Directory — **mkvob** also creates a special directory element, **lost+found**, as a subdirectory of the VOB root directory. (Only dynamic views can access **lost+found**.) ClearCase and Attache place elements that are no longer cataloged in any directory version in this directory. This occurs when you do any of the following:

- Create new elements, and then **uncheckout** the directory in which they were created
- Delete the last reference to an element with the **rmname** command
- Delete the last reference to an element by deleting a directory version with the **rmver**, **rmbranch**, or **rmelem** command

When an element is moved to **lost+found**, it gets a name of the form

element_leaf_name.id-number

The id-number is a unique hexadecimal number, such as

```
41a00000bcaa11caacd0080069021c7
```

The **lost+found** directory has several unique properties:

- It cannot be checked out.
- It can be modified.
- No branches can be created within it.

To move an element from the **lost+found** directory to another directory within the VOB, use the **cleartool mv** command. To move an element from the **lost+found** directory to another VOB, use the **relocate** command.

To conserve disk space, periodically clean up the **lost+found** directory:

1. If you need an element in **lost+found**, catalog it in a versioned directory using **mv**.
2. Use **rmelem** command to remove unneeded elements.

VOB TEXT MODES

Each VOB can operate in either of two modes:

- **Standard Mode** — In this mode, no special provision is made in the VOB for correctly reporting the sizes of text files stored in views created with **mkview -tmode msdos**. All VOBs are initially created in this mode.
- **MS-DOS-Enabled Mode** — In this mode, the VOB database tracks the number of text lines in each version of each text file. A utility program, **msdostext_mode**, places a VOB in this mode (or restores it to standard mode).

See *Administering ClearCase* for detailed information on MS-DOS text mode.

VOB REGISTRY AND VOB ACTIVATION

Each VOB is registered in the network wide storage registry, as described in the **registry_ccase** reference page. The **mount** command activates a registered VOB by mounting it as a type-MVFS file system (ClearCase and Attache dynamic views only). See the (**cleartool**) **mount** reference page for details.

SEE ALSO

chpool, **config_spec**, **events_ccase**, **lsvob**, **mkvob**, **mkdir**, **mkelem**, **mkpool**, **mount**, **mvfsstorage**, **protectvob**, **registry_ccase**, **schedule**, **scrubber**, **view**, **vob_scrubber**

vob_restore

Restores a VOB from backup media.

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command
Attache	command

SYNOPSIS

ccase-home-dir/**etc/vob_restore** [**–restart** *restart-path*] *vob-tag*

DESCRIPTION

The **vob_restore** command restore a damaged VOB. It prompts for all required input and displays explanatory text at each step. If you quit **vob_restore** before it completes, you can use the **–restart** option to resume VOB restoration at the point where you stopped it.

NOTE: We strongly recommended that you do not retrieve VOB storage from backup until prompted to do so by **vob_restore**. If you get the VOB storage directory from backup media before running **vob_restore**, you must either unregister the VOB before retrieving the backup or stop ClearCase or ClearCase LT before retrieval and restart it afterward. If you wait until prompted, **vob_restore** performs the necessary steps, safeguarding the integrity of your restored VOB.

Restoring a VOB from Separate Database Snapshot and VOB Storage Directories

If the VOB to be restored is backed up using the scenario described in **vob_snapshot**, the most comprehensive recovery operation requires you to do the following:

- Retrieve a damaged VOB's storage directory from backup media and load it into a temporary disk storage area
- Retrieve the damaged VOB's database from backup media and/or point to the on-disk snapshot location

In this case, **vob_restore** does all of the following:

- Runs the **db_check** utility on the database snapshot, if the snapshot was taken from an unlocked VOB and the **db_check** was not run at snapshot time
- Copies the snapshot directory to a temporary storage location, overwriting the VOB database that was (most likely) restored from the VOB storage directory backup

- Unregisters the VOB from the local host's network region
- Stops ClearCase or ClearCase LT on the local host
- Prompts you to move or remove the VOB storage at the original storage location, if it exists
- Copies the restored VOB storage directory from its temporary storage location to the VOB's registered location
- (Optional) Runs **checkvob** to find and fix inconsistencies between the data containers in VOB storage pools and the VOB database
- (ClearCase MultiSite only) If the VOB is replicated, indicates that you should run **restorereplica**

If You Are Not Restoring a VOB Database Snapshot

If the damaged VOB is not currently being processed by **vob_snapshot**, do not supply a snapshot directory when prompted for one. A VOB database was backed up as part of the VOB storage directory backup, and you will restore the entire VOB as a single unit.

In this case, if the VOB was backed up while unlocked, **vob_restore** forces a potentially time-consuming **db_check** operation on the VOB database.

If There Is No Need to Preserve the Damaged Storage Area

If the VOB storage directory is irretrievably missing or damaged, or if you do not have sufficient disk space for two copies of the VOB storage directory, you can copy the VOB storage backup directly to the VOB's registered storage location. Do not supply a temporary storage location when prompted for one.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the *root* user to execute this command.

Locks: If the VOB is still accessible, lock it to prevent any further changes. **vob_restore** leaves the VOB locked when it completes.

OPTIONS AND ARGUMENTS

If you do not specify all required command line arguments, **vob_restore** prompts for input.

SPECIFYING THE VOB. *Default:* None. You must supply a VOB-tag. **vob_restore** prompts for all additional information.

vob-tag
The VOB's VOB-tag, as specified in **mkvob** or **mktag –vob**.

SPECIFYING A RESTART FILE. *Default:* None. If you omit this option, **vob_restore** does not attempt to find a restart-state file that may have resulted from an earlier, aborted **vob_restore** invocation on the same VOB-tag.

–restart *restart-path*
Specifies the pathname of the restart file saved during a previous invocation. Always record the *restart-path* that is reported by **vob_restore** when you stop VOB restoration before it has completed.

SAMPLE RECOVER PROCEDURE

This sample recover procedure merges a VOB database snapshot and a VOB storage directory backup to restore the damaged VOB **/vobs/src**.

1. Log in to the VOB host as **root**.
2. As a precaution, lock the VOB, so that no further changes can be made. Note that read access can continue if the VOB is in a readable state (damaged database only, for example).

   cleartool lock –c "vob recover required" vob:/vobs/src
3. Retrieve the most recent VOB database snapshot and storage directory from backup media.
   a. Retrieve the most recent VOB database snapshot directory, **/net/saturn/usr1/snaps/vob_src**.

      NOTE: The most recent snapshot may still be on disk, or it may be on backup media.
   b. Use **cleartool lsvob /vobs/src** to identify the storage directory pathname:

      cleartool lsvob /vobs/src
      ```
      /vobs/src      /net/io/vobstore/src.vbs
      ```
   c. Retrieve the VOB storage directory from backup media, and place it in temporary storage location **/net/io/vobstore/temp.vbs**. (If no users are reading the damaged VOB, and it is presumed destroyed, you may choose to restore the VOB storage backup directly to its registered location, rather than to a temporary area.)
4. Run **vob_restore** to merge the VOB database and storage directory:

   *ccase-home-dir***/etc/vob_restore /vobs/src**

   vob_restore prompts for all necessary input and displays explanatory text.
5. Restart ClearCase or ClearCase LT on the local host.
6. Re-register **/vobs/src**:

   cleartool mktag –vob –tag /vobs/src /net/io/vobstore/src.vbs

   cleartool register –vob /net/io/vobstore/src.vbs

EXAMPLES

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form

/vobs/*vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

- Restore VOB **/vobs/src**. To complete the recovery, run **checkvob** to find and fix inconsistencies between the restored VOB database and the restored VOB storage pools. **vob_restore** prompts for all required information.

  NOTE: In this example, as is recommended, the restored data is not retrieved from backup media before running **vob_restore**.

  /usr/atria/etc/vob_restore /vobs/src

SEE ALSO

checkvob, **vob_snapshot**, **vob_snapshot_setup**

vob_scrubber

Remove event records and oplog entries from VOB database

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

ccase-home-dir/**etc/vob_scrubber** [**–stats_only**] [**–long**] [**–nlog**]
{ **–lvobs** | *vob-storage-dir-pname* ... }

DESCRIPTION

The **vob_scrubber** program deletes old event records and MultiSite *oplog entries* from a VOB database. This retards VOB growth by logically deleting the items, freeing space in the VOB database for storage of new event records and oplog entries. (Physical deletion requires processing with the **reformatvob** command.)

vob_scrubber does not need to run in a view and does not require the VOB it processes to be mounted.

CLEARCASE AND ClearCase LT EVENTS

ClearCase and ClearCase LT create a meta-data item called an event record in a VOB database almost every time it modifies the database—for example, to record the checkin of a new version, the attaching of an attribute to an element, or the creation of a new branch type. Each event record consumes 300–400 bytes. Some event records, like those for element and version creation, are valuable indefinitely; however, many minor event records are not. For example, the removal of a version label from a collection of versions creates a minor event record for each affected object. Over time, such minor event records occupy more space as they become less useful. (After a month or a year, no one is likely to care who removed the version labels, especially if the label type itself has also been deleted.)

Event Record Scrubbing

vob_scrubber marks certain event records as logically deleted. As with any metadata removal, the deletion does not physically reduce the amount of disk space used by the VOB database; it merely frees up space in the database, making it available for future use. To actually reduce the size of the database, you must run **reformatvob**, which discards the logically deleted data as it reconstructs the VOB database. Thus, regular use of **vob_scrubber** minimizes VOB database growth, but does not recover disk space.

What Event Records Are Deleted

These obsolete event records are always deleted, regardless of scrubbing parameters:

- Creation event records for derived objects.
- Event records whose operations are **mkattr**, **mkhlink**, **mklabel**, **mktrigger**, **rmlabel**, **rmhlink**, **rmattr**, or **rmtrigger** (if the type object associated with the event has been deleted with **rmtype**).

These event records are never deleted:

- The most recent 1000 event records physically added to the VOB (regardless of logical event time). These are needed by views for cache invalidation.
- The most recent lock event record (for an object that is locked).
- Event records for operations not annotated with an **S** in Table 4 in the **events_ccase** reference page.

All other event records are preserved or deleted according to the configuration file specifications described in *VOB-SPECIFIC EVENT-RECORD SCRUBBING PARAMETERS*.

MULTISITE OPLOG ENTRIES

In each replicated VOB, **cleartool** creates *oplog* (operation log) entries, which store all the information required to repeat the changes in some other replica of the VOB.

Each oplog entry logically includes this information:

- The identity of the replica where the change originally took place.
- The VOB-database-level change—for example, creation of a new element, checkin of a new version, attaching of an attribute, and so on.
- The storage-pool-level change, if any—for example, the contents of a new version.
- The event record generated for the change.
- An integer sequence number—**1** for the first change originating at a particular replica, **2** for the next change, and so on. This is the *epoch number* of the oplog entry.

Like event records, oplog entries are stored in the VOB database and can be scrubbed.

WARNING: Oplog entries play an essential role in the MultiSite replica-synchronization scheme. It is extremely important that oplog entries be retained until they are no longer needed for synchronization. For this reason, the standard retention period for oplog entries is infinite (**oplog –keep forever**).

MULTISITE EXPORT_SYNC ENTRIES

When you export an update packet from a replicated VOB, MultiSite creates one export_sync record for each target replica. These records are stored in the VOB database and are used by the **recoverpacket** command to reset a replica's epoch matrix. You can scrub these records, but scrubbing old records limits the date range over which **recoverpacket** can operate.

NOTE: export_sync records are distinct from exportsync events, which are listed by the **lshistory** command and graphical History Browser. You can scrub export_sync records without losing export history for a replica.

AUTOMATIC SCRUBBING

By default, the scheduler runs **vob_scrubber** periodically. See the **schedule** reference page for information on describing and changing scheduled jobs. A configuration file, **vob_scrubber_params**, provides control over which event records and oplog entries are deleted.

You can run **vob_scrubber** manually as needed.

OPTIONS AND ARGUMENTS

REPORT FORMAT. *Default:* Event statistics are listed briefly, categorized by kind of object. For example, all event records for branch objects are grouped.

–long
Produces a detailed report of the event statistics, categorized by kind of object, kind of event, and kind of operation.

REPORT DESTINATION. *Default:* The report is sent to the standard log file, **/var/adm/atria/log/vob_scrubber_log**.

–nlog
Sends the report to **stdout** instead of the log file.

DELETION CONTROL. *Default:* Delete event/oplog records and report statistics on the number of objects, the number of records before deletion, the number of records deleted, and the number of records after deletion.

–stats_only
Suppresses deletion of records; the report includes statistics on the number of objects, event records, and oplog entries in the VOB.

VOBS TO BE PROCESSED. *Default:* None.

–lvobs
Scrubs event records and oplog entries from all mounted VOBs that reside on the local host.

vob-storage-dir-pname
Scrubs the VOB whose storage directory is at the specified pathname.

VOB-SPECIFIC EVENT-RECORD SCRUBBING PARAMETERS

A host-wide configuration file controls the operation of **vob_scrubber**; each VOB can have its own configuration file, which overrides the systemwide settings:

Host-wide config file	*ccase-home-dir*/**config/vob/vob_scrubber_params**
Per-VOB config file	*vob-storage-dir-pname*/**vob_scrubber_params**

The event-scrubbing configuration file is a text file. A line that begins with a number sign (**#**) is a comment. All other lines control how one kind of event is to be scrubbed—how long to keep the most recent one, and how long to keep other events of that kind:

event *operation* **–keep_all** { *n* | **forever** } [**–keep_last** { *n* | **forever** }]

These are the components of an event-scrubbing control line:

event
: A keyword that indicates that the remaining components of the control line apply to the event records created by a particular CM operation. (See the **events_ccase** reference page for a list of operations and the associated object to which event records are attached.)

operation
: Kind of event, specified by the operation that creates the event record. (See the **events_ccase** reference page for a list of operations and the associated objects to which event records are attached.)

–keep_all { *n* | **forever** }
: For each object: keep event records created by the specified operation for at least *n* days, or forever. If **–keep_last** is also specified, this period applies to all but the most recent such event; otherwise, the period applies to all such events, including the most recent one.

–keep_last { *n* | **forever** }
: (Optional) For each object: keep the most recent event record created by the operation for at least *n* days, or forever. The **keep_last** period must be at least as long as the **keep_all** period. The meaning of "most recent event" depends on the operation; see the **events_ccase** reference page for a list of operations and the associated objects to which event records are attached.

OPERATION LOG AND EXPORT RECORD SCRUBBING

The **vob_scrubber_params** files also control scrubbing of oplog entries and export records. The syntax for these lines follows. (Do not begin these lines with the keyword **event**.)

oplog –keep { *n* | **forever** }
: Specifies the number of days an oplog entry is kept in the VOB database. You must preserve oplog entries long enough to guarantee delivery of synchronization updates based on them. The default is **forever**.

export_sync –keep { *n* | **forever** }
Specifies number of days an export synchronization record is kept in the VOB database. By default, this line is not included in the **vob_scrubber_params** file, and the records are scrubbed with the same frequency as the oplog entries.

SCRUBBING DEFAULTS

If the configuration file includes no control line for a particular operation, all event records created by the operation are kept forever. Therefore, an empty configuration file preserves all event records (except obsolete ones, which are always discarded; see *What Event Records Are Deleted*). The calculated times are always compared to the logical event creation time (as shown by **lshistory**), rather than the physical event creation time. These can differ if the event records were created by an exporter, such as **clearexport_pvcs**.

If the configuration file includes no **–oplog** control line, then oplog entries are kept forever.

EXAMPLES

- For **unlock** events in all VOBs on the local host: keep the event record if the event occurred within the past 7 days (but keep an event that occurred within the past 30 days if it is the most recent event on a particular object). Otherwise, delete the event record.

  In '*ccase-home-dir*/**config/vob/vob_scrubber_params**':

  ```
  event unlock -keep_all 7 -keep_last 30
  ```

- In the VOB replica whose storage directory is **/vobstore/tromba.vbs**, retain oplog entries for a year.

  In '**/vobstore/tromba.vbs/vob_scrubber_params**':

  ```
  oplog -keep 365
  ```

FILES

ccase-home-dir/**config/vob/vob_scrubber_params**
/var/adm/atria/log/vob_scrubber_log

SEE ALSO

reformatvob, **lshistory**, **events_ccase**, **scrubber**, **schedule**

vob_server

Server program for VOB storage pool access

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

Invoked as needed by the **albd_server** program

DESCRIPTION

For each VOB, a long-lived **vob_server** process runs on the VOB host, with the user-ID of the VOB owner (see **protectvob**). This process maintains the VOB's storage pools in response to requests from client processes. This includes creating, deleting, and controlling the UNIX-level permissions of the pools' data containers.

The **vob_server** is the only process that ever creates or deletes data containers; the VOB owner is the only user who can modify data containers and storage pools. These severe restrictions protect VOB data against careless or malicious users.

A **vob_server** process is started as needed by **albd_server**. It remains active until the operating system is restarted, the VOB is deleted with the **rmvob** command, or a ClearCase or ClearCase LT shutdown is performed (see the **init_ccase** reference page).

CONFIGURATION FILE

The configuration file for the **vob_server** is named **vob_server.conf** and is stored in the VOB storage directory. **vob_server.conf** is a text file and is read when the **vob_server** starts up. This file contains the setting for deferred deletion of source containers. Deferred deletion is activated if you have enabled VOB database snapshot activity for the VOB with the **vob_snapshot_setup** utility.

Deferred Source Container Deletion

Deferred deletion ensures that the source pool will contain all needed containers when a backup program archives an active VOB. When a container is replaced by new version data (for example, during a checkin), the new container is created and the client requests the **vob_server** to remove the old container. If deferred deletion is deactivated, the container is immediately removed. If deferred deletion is enabled, the old container is added to a list of pending deletions, and it is removed in 30 minutes. Keeping source data containers for 30 minutes increases disk space

requirements. The increase can be substantial during any 30-minute interval of heavy VOB checkin activity.

You can run **kill –HUP** on the **vob_server** process to send deferred deletion statistics to the **vob_server** log (see also **getlog** and **getlog –gr**).

When **checkvob** examines source pools, it reports containers on the deferred deletion list.

The deferred deletion list is written every five minutes to the file **delete_list.db** in the VOB storage directory. To force deletion of all containers on the deferred list:

1. Edit **vob_server.conf** and deactivate deferred deletion.
2. Wait five minutes.
3. Edit **vob_server.conf** and reenable deferred deletion.

ERROR LOG

The **vob_server** process sends warning and error messages to **/var/adm/atria/log/vob_log**.

SEE ALSO

albd_server, **checkvob**, **db_server**, **nfsd(1M)**, **vob**, **vobrpc_server**

vob_snapshot

Copies the VOB databases of all local VOBs or replicas configured for database snapshot

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command
Attache	command

SYNOPSIS

ccase-home-dir/**etc/vob_snapshot**

DESCRIPTION

The **vob_snapshot** command makes an on-disk copy of a local, locked VOB database. Using this command reduces the amount of time a VOB database needs to be locked when you back up the VOB. Later, as part of your standard system backup procedure, the VOB storage directory (minus the VOB database directory) and the VOB database snapshot can be backed up without locking the VOB. Because the database snapshot and VOB storage pool backups occur at different times, they are likely to be slightly out of sync. To correct this skew, the **checkvob** utility resynchronizes the VOB database and storage pools when you run **vob_restore**.

By default, the scheduler runs **vob_snapshot** periodically. See the **schedule** reference page for information on describing and changing scheduled jobs. If no locally stored VOBs are configured for database snapshot, **vob_snapshot** exits silently.

A local VOB's database is copied only if snapshot parameters have been applied to it with the **vob_snapshot_setup** utility. See *Per VOB (or Replica) Snapshot Parameters*.

Per Host Snapshot Parameters

The file **/var/adm/atria/config/snapshot.conf** stores information used to notify interested parties of VOB database snapshot activity on a particular host. Here are the parameters in **snapshot.conf** and their default values:

```
NOTIFICATION_PROGRAM=/usr/atria/bin/notify
NOTIFICATION_LIST=root
CONFIRMATION_ON_SUCCESS=yes
```

Per VOB (or Replica) Snapshot Parameters

When **vob_snapshot** runs on a VOB host, it checks each locally stored VOB for the existence of a multipart string attribute that specifies snapshot parameters. An administrator uses the

vob_snapshot_setup utility to apply the **vob_snapshot_parameters** attribute to each VOB or replica for which snapshots will be taken. The attribute string's individual components specify the following:

- Where to put the VOB database snapshot

  A disk location that will store the snapshot. Typically, this location gets backed up later (along with the VOB storage directory as part of normal backup operations), and it is overwritten by the next snapshot.

- Whether to run **db_check** on the VOB database snapshot

  The **db_check** utility performs fundamental database consistency and integrity checks. (Later, at recover time, **checkvob** may examine the VOB database looking for ClearCase or ClearCase LT anomalies.) The **db_check** pass occurs after all snapshots are complete on the local host.

- Which users should be notified (receive mail) about snapshot operations on this VOB

  A list of user names to be notified when **vob_snapshot** processes this VOB. This per-VOB list supplements the per-host notification list maintained in **/var/adm/atria/config/snapshot.conf**. The **snapshot.conf** file also specifies the notification program to be used.

See **vob_snapshot_setup** for more information on setting these parameters for a particular VOB.

Database Snapshot Details

When **vob_snapshot** encounters a VOB that is configured for database snapshot, it performs the following steps (logging messages in the **snapshot_log** file along the way):

1. Verifies that the snapshot target directory exists and is writable.
2. Locks the VOB. If **vob_snapshot** cannot lock the VOB, it proceeds with the snapshot, but logs the snapshot's status as questionable.
3. Checks the VOB's specified snapshot target directory for sufficient disk space.
4. Creates a subdirectory whose name is the VOB's replica UUID. If a directory with that name already exists, remove it first (that is, remove the previous snapshot).
5. Copies the VOB database directory tree (using **tar**) to the subdirectory created in Step #3.
6. Unlocks the VOB.
7. Repeats Step #1 through Step #6 for the next VOB.
8. Runs **db_check** on all VOBs configured for this check.

   NOTE: If the log or notification mail reveals a failed **db_check**, check the log for obvious errors. If you cannot resolve the problem, contact Rational Technical Support.
9. Sends mail to per VOB and per host notification lists.

If You Do Not Use vob_snapshot

You must lock the VOB and back up the entire VOB storage directory (and any remote storage pools). Such a backup avoids the issue of skew between VOB database snapshot and VOB storage pools (which are typically backed up some time after the snapshot), but it requires that the VOB remain locked during the entire backup operation.

PERMISSIONS AND LOCKS

Permissions Checking: You must be the *root* user. See the **permissions** reference page.

Locks: The VOB must be locked to guarantee the integrity of a database snapshot. If **vob_snapshot** cannot lock the VOB (because it is run with insufficient permissions, or another user locked the VOB), it proceeds with the copy operation but logs the snapshot's status as "questionable." This status is upgraded to "successful" if the optional post-snapshot **db_check** pass succeeds.

OPTIONS AND ARGUMENTS

None.

EXAMPLE VOB BACKUP AND RECOVER SCENARIO USING vob_snapshot

This sample backup/recover scenario uses the utilities **vob_snapshot_setup**, **vob_snapshot**, and **vob_restore**.

1. Log in to the VOB host as the *root* user.
2. Configure the VOB **/vobs/src** for daily database snapshots:

   /usr/atria/etc/vob_snapshot_setup modparam –dbcheck yes –notify root,clearadm,anne –snap_to /net/saturn/usr1/snaps/vob_src /vobs/src

   The VOB database is copied to a subdirectory of the **–snap_to** directory you specify. The subdirectory name matches the VOB's *replica UUID* (which you can display with **vob_snapshot_setup lsvob –long**). In this example, the snapshot is stored in **.../snaps/vob_src/***replica-uuid*.

   NOTE: If run without command line options, **vob_snapshot_setup modparam** prompts for all necessary input and displays explanatory text.
3. Confirm that **/vobs/src** is in the local host's snapshot list:

   /usr/atria/etc/vob_snapshot_setup lsvob

   ```
   /vobs/src
   ...
   ```
4. Make sure the **–snap_to** directory **/net/saturn/usr1/snaps/vob_src** and the VOB storage directory are saved by your normal backup routine.

NOTE: Although **/vobs/src** must be locked at database snapshot time, it need not be locked when your normal backup routine saves the snapshot and VOB storage directory.

5. Let VOB **/vobs/src** be locked and its database copied to disk by the standard **vob_snapshot** job run by the scheduler. Also, let your backup procedure capture the VOB storage directory and VOB database snapshot.

   A common routine is to run **vob_snapshot** early in the morning or at lunch time, when the VOB lock is least likely to interfere with nightly builds, and to back up VOB storage directories at night.

(End backup scenario)

6. **/vobs/src** fails (corrupt or lost database, corrupt or lost storage pools, and so on).

(Begin recover scenario)

7. Log in to the VOB host as the *root* user.
8. As a precaution, lock the VOB, so that no further changes can be made. Note that read access can continue if the VOB is in a readable state (damaged database only, for example).

   cmd-context **lock –c "vob recover required" vob:/vobs/src**
9. Retrieve the most recent VOB database snapshot and storage directory from backup media.
   a. Retrieve the most recent VOB database snapshot directory, **/net/saturn/usr1/snaps/vob_src**.

      NOTE: The most recent snapshot may still be on disk, or it may be on backup media.

   b. Use **lsvob /vobs/src** to identify the storage directory pathname:

      cmd-context **lsvob /vobs/src**

      ```
      /vobs/src      /net/io/vobstore/src.vbs
      ```

   c. Retrieve the VOB storage directory from backup media, and place it in temporary storage location **/net/io/vobstore/temp.vbs**. (If no users are reading the damaged VOB, and it is presumed destroyed, you may choose to restore the VOB storage backup directly to its registered location, rather than to a temporary area.)
10. Run **vob_restore** to merge the VOB database and storage directory:

    /usr/atria/etc/vob_restore /vobs/src

    NOTE: If run without command line options, **vob_restore** prompts for all necessary input and displays explanatory text.

    vob_restore does the following:

    - Moves the snapshot directory to a temporary storage location, overwriting the VOB database that was (most likely) restored from the VOB storage directory backup

- Unregisters **/vobs/src** from the local host's network region
- Stops ClearCase or ClearCase LT on the local host
- Prompts you to move or remove the VOB storage at the original storage location, if it exists
- Copies the restored VOB storage directory from its temporary storage location to the VOB's registered location
- (Optional) Runs **checkvob** to find and fix inconsistencies between the data containers in VOB storage pools and the VOB database

11. Restart ClearCase or ClearCase LT on the local host.

12. Re-register **/vobs/src**:

cmd-context **mktag –vob –tag /vobs/src /net/io/vobstore/src.vbs**

cmd-context **register –vob /net/io/vobstore/src.vbs**

End recover scenario

See also **vob_snapshot_setup** and **vob_restore**.

THE SNAPSHOT LOG FILE

vob_snapshot's operation is recorded in the log file **/var/adm/atria/log/snapshot_log**. If snapshot notification mail indicates a problem with any snapshot, consult the **snapshot_log** file. Here are some common error and status messages from **snapshot_log**:

Successful snapshot messages:

```
NOTE: VobTag: /vobs/src
NOTE: Replica UUID: 0b9ccb24.8dc211cf.af59.00:01:80:73:db:6f
NOTE: Family UUID: 0b9ccb20.8dc211cf.af59.00:01:80:73:db:6f
NOTE: Dbcheck succeeded
NOTE: SNAPSHOT COMPLETED SUCCESSFULLY
```

The following messages may be generated when **vob_snapshot** runs on an unlocked VOB. If the database check passes, the snapshot is upgraded from "questionable" to "successful." You should always lock a VOB before copying its VOB database with **vob_snapshot**.

```
ERROR: Could not lock the replica for the copy. Copied anyway.
ERROR: The snap was done without the vob lock in place.
```

```
ERROR: SNAPSHOT QUESTIONABLE
NOTE : The snap was done unlocked but the database checked ok
```

The following error occurs if the user account under which **vob_snapshot** executes does not have permission to overwrite the directory supplied as a **–snap_to** argument to **vob_snapshot_setup**:

```
ERROR: The snap_to directory for vob /vobs/src is not writable
```

EXAMPLES

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

- List all VOBs on the local host's snapshot list (VOBs to which snapshot parameters have been applied with **vob_snapshot_setup modparam**). Then, take VOB database snapshots for all VOBs in the list.

  /usr/atria/etc/vob_snapshot_setup lsvob

  ```
  /vobs/src
  /vobs/lib
  ```

  /usr/atria/etc/vob_snapshot

- Add local VOB **/vobs/proj1** to the current host's snapshot list. Then, take VOB database snapshots for all VOBs in the list.

  /usr/atria/etc/vob_snapshot_setup modparam –dbcheck yes –notify root,clearadm,anne –snap_to /net/saturn/usr1/snaps/proj1 /vobs/proj1

  /usr/atria/etc/vob_snapshot

  Sample notification mail for snapshot of **/vobs/proj1**:

  ```
  From root Fri Apr  5 22:12:01 1999
  To: clearadm,root,anne
  Subject: Snapshot Status

  NOTE    : ++++++++++++++++++++++++++++++++++++++++++++++++++
  NOTE    : VobTag: /vobs/proj1
  NOTE    : Hostname: saturn
  NOTE    : Replica UUID: 4a6bbe5d.88d511cf.a9b4.00:01:80:73:db:6f
  NOTE    : Family UUID: 4a6bbe59.88d511cf.a9b4.00:01:80:73:db:6f
  NOTE    : Dbcheck succeeded
  NOTE    : SNAPSHOT COMPLETED SUCCESSFULLY
  NOTE    : ++++++++++++++++++++++++++++++++++++++++++++++++++
  ```

FILES

/var/adm/atria/config/snapshot.conf
/var/adm/atria/log/snapshot_log

SEE ALSO

checkvob, **schedule**, **vob_restore**, **vob_snapshot_setup**

vob_snapshot_setup

Sets or displays VOB database snapshot parameters

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command
Attache	command

SYNOPSIS

- List local VOBs that are currently processed by **vob_snapshot**:

  *ccase-home-dir***/etc/vob_snapshot_setup lsvob** [**–short** | **–long**] [*vob-tag* ...]

- Configure a VOB for **vob_snapshot** processing:

  *ccase-home-dir***/etc/vob_snapshot_setup modparam –snap_to** *snap-dir-pname*
  { **–dbcheck yes** | **–dbcheck no** } **–notify** *login-name*[**,**...] *vob-tag*

- Remove a VOB from the database snapshot list:

  *ccase-home-dir***/etc/vob_snapshot_setup rmparam** *vob-tag*

DESCRIPTION

Use **vob_snapshot_setup** to control VOB database snapshot activity on each VOB host. By default, the scheduler runs **vob_snapshot** periodically. When **vob_snapshot** runs on a VOB host, it checks each locally stored VOB for the existence of a multipart string attribute that specifies snapshot parameters. A VOB's database is copied by **vob_snapshot** only if this attribute has been applied to the VOB with **vob_snapshot_setup**.

Use **vob_snapshot_setup lsvob** to list the local VOBs currently processed by **vob_snapshot** and, with **–long**, to display the snapshot parameters for each VOB in the list.

Use **vob_snapshot_setup modparam** to add a VOB to the database snapshot list, or to change snapshot parameters for a VOB already on the list. (You cannot modify individual parameters with **modparam**, but must replace them all.)

Use **vob_snapshot_setup rmparam** to remove a VOB from the snapshot list.

See also **vob_snapshot** and **vob_restore**.

WARNING: If you are using a homegrown or third-party type manager, code that implements the **get_cont_info** method must be added to the type manager, or elements managed by the type manager cannot be processed by **checkvob** at **vob_restore** time.

Setting VOB Snapshot Parameters

An administrator uses **vob_snapshot_setup modparam** to apply the following snapshot parameters to each VOB or replica for which database snapshots are to be taken:

Parameter	Legal Values	Default Value
–snap_to	existing, writable directory pathname	no default
–dbcheck	yes | no	no
–notify	comma-separated list	empty list

These parameters are combined to form a single string attribute of type **vob_snapshot_parameters**, which **vob_snapshot_setup** attaches to the VOB.

Here is how these parameters may appear in a **vob_snapshot_setup lsvob –long** listing:

```
VobTag:                 /vobs/src
...
Dbcheck Enabled:        yes
Notification List:      root user,clearadm,anne
Snap To:                /net/saturn/usr1/snapshots...
```

See the **–snap_to**, **–dbcheck**, and **–notify** options for further details.

Disk Space Usage

The VOB snapshot backup/restore scenario requires additional disk space, both at restore time and during daily operation:

- At restore time, **checkvob** may require substantial disk space. See the **checkvob** reference page.
- Enabling VOB snapshots for a VOB also enables a deferred source container deletion mechanism, which typically increases source pool size. See **vob_server** for a description of deferred deletion.

PERMISSIONS AND LOCKS

Permissions Checking: You must be **root** or the VOB owner to execute this command. See the **permissions** reference page.

Locks: A **modparam** or **rmparam** operation fails if any of the following objects are locked: VOB, **vob_snapshot_parameters** attribute type.

Mastership: If the VOB is replicated, it must be self-mastering.

OPTIONS AND ARGUMENTS

VOB LISTING REPORT FORMAT. *Default:* **vob_snapshot_setup lsvob** lists the VOB-tag of each local VOB currently configured for database snapshot.

–short
Same as default.

–long
In addition to the VOB-tag, **vob_snapshot_setup lsvob** lists each VOB's snapshot parameters and additional VOB identity details. If multiple VOBs use the same parent **–snap_to** directory, use the replica UUID returned by **–long** to find a particular snapshot in the parent directory.

vob-tag ...
Space-separated list of VOBs; restricts listing to one or more local VOBs.

SPECIFYING THE VOB. *Default:* None. **modparam** and **rmparam** operations require a VOB-tag argument.

vob-tag
The VOB's VOB-tag, as specified in **mkvob** or **mktag –vob**.

SETTING SNAPSHOT PARAMETERS. *Default:* With **modparam**, you must specify a VOB-tag; if you specify no other options or arguments, **modparam** prompts for all necessary input and displays explanatory text. If you specify both a VOB-tag and a snapshot target directory, **modparam** does not prompt for additional parameters: **vob_snapshot** does not run the **db_check** operation, and the notify list is empty.

–snap_to *snap-dir-pname*
A disk location to store the snapshot. **vob_snapshot** appends the VOB's replica UUID to the **–snap_to** directory to create a subdirectory, then copies the VOB database to the subdirectory (after checking for sufficient disk space).

The replica UUID subdirectory that stores a VOB's database snapshot is overwritten the next time **vob_snapshot** processes that VOB.

Typically, the **–snap_to** directory gets backed up as part of normal backup operations some time after the snapshots are taken.

–dbcheck yes
–dbcheck no
Specifies whether to run the **db_check** utility on each snapshot.

vob_snapshot runs **db_check** to perform fundamental database consistency and integrity checks. (Later, at restore time, **checkvob** may examine the VOB database looking for ClearCase and ClearCase LT anomalies.) The **db_check** pass occurs after all

snapshots are completed on the local host. Because this check can be time-consuming, it is disabled by default.

If **vob_snapshot** cannot lock the database and **db_check** is disabled, **db_check** runs on the snapshot at **vob_restore** time. Running **db_check** earlier, at snapshot time, may expose problems you would prefer not to encounter at recover time.

–notify *login-name*[**,**...]
A list of user-IDs to be notified when **vob_snapshot** processes this VOB or replica. This VOB-specific list supplements the per host notification list maintained in **/var/adm/atria/config/snapshot.conf**. The **snapshot.conf** file also specifies the notification program to be used. If you do not want to supply a list of user IDs to be notified, specify **–notify """"** on the command line.

EXAMPLES

NOTE: In the examples that follow, arguments and output that show multicomponent VOB tags are not applicable to ClearCase LT, which recognizes only single-component VOB tags. In this manual, a multicomponent VOB tag is by convention a two-component VOB tag of the form **/vobs/***vob-tag-leaf*—for example, **/vobs/src**. A single-component VOB tag consists of a leaf only—for example, **/src**. In all other respects, the examples are valid for ClearCase LT.

- List all VOBs on the local host that are currently configured for processing by **vob_snapshot**.

  /usr/atria/etc/vob_snapshot_setup lsvob
  ```
  /vobs/src
  /vobs/lib
  ```

- Same as previous example, but expand the output to include each VOB's replica UUID and snapshot parameters.

  /usr/atria/etc/vob_snapshot_setup lsvob –long
  ```
  VobTag:                 /vobs/src
  Replica Name:           original
  Replica Uuid:           4a6bbe5d88d511cfa9b400018073db6f
  Family  Uuid:           4a6bbe5988d511cfa9b400018073db6f
  Dbcheck Enabled:        yes
  Notification List:      root,clearadm,anne
  Snap To:                /net/saturn/usr1/snapshots
  Deferred Deletes:       Enabled
  ```

```
VobTag:                 /vobs/lib
Replica Name:           original
Replica Uuid:           5fec90f48db911cfab9800018073db6f
Family  Uuid:           5fec90f08db911cfab9800018073db6f
Dbcheck Enabled:        no
Notification List:      root,clearadm,bill
Snap To:                /net/saturn/usr1/snapshots
Deferred Deletes:       Enabled
```

- Add VOB **/vobs/src** to the local host's snapshot list.

  /usr/atria/etc/vob_snapshot_setup modparam –dbcheck yes –notify root,clearadm,anne –snap_to /net/saturn/usr1/snaps/src /vobs/src

- Add **/vobs/src** to the local host's snapshot list, as in the previous example, but this time run **vob_snapshot_setup modparam** in interactive mode.

  vob_snapshot_setup modparam /vob/src

  ```
  Supply a directory to contain the snapshot data for this vob. The
  directory must exist and be writable.
  (Full pathname: there is no default) /net/saturn/usr1/snaps/src
  ```

  ```
  Supply a comma separated list of those users to be notified of events
  during the snapshot of this vob.
  (Comma separated user id list: default no one): root,clearadm,anne
  ```

  ```
  Do you want a data base check to be performed on this vob after all
  snapshot operations on this host are completed?
  Valid responses are (yes,no)
  The default is no: yes
  ```

- Remove VOB **/vobs/src** from the local host's snapshot list.

  /usr/atria/etc/vob_snapshot_setup rmparam /vobs/src

FILES

/var/adm/atria/config/snapshot.conf
/var/adm/atria/log/snapshot_log

SEE ALSO

checkvob, **schedule**, **vob_restore**, **vob_snapshot**

vobrpc_server

ClearCase database server program

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

Invoked as needed by the **albd_server** program

DESCRIPTION

Each VOB host runs up to five **vobrpc_server** processes for each of its VOBs. Each process handles requests from **view_server** processes throughout the network. The request can generate both metadata (VOB database) and file-system data (storage pool) activity. The **vobrpc_server** accesses the VOB database in exactly the same way as a **db_server**. It forwards storage pool access requests to the **vob_server**.

Multiple server processes are started by **albd_server**, which also routes new requests to the least-busy servers and terminates unneeded **vobrpc_server** processes when the system is lightly loaded.

ERROR LOG

The **vobrpc_server** process sends warning and error messages to **/var/adm/atria/log/vobrpc_server_log**.

SEE ALSO

albd_server, **db_server**, **init_ccase**, **nfsd(1M)**, **vob**, **vob_server**

wildcards

Pattern-matching characters for Attache pathnames

APPLICABILITY

Product	Command Type
Attache	general information

NOTE: For ClearCase and ELCC, see the **wildcards_ccase** reference page.

SYNOPSIS

? ***** **~** **[...]** **...** **~***username*

DESCRIPTION

Attache recognizes wildcard (pattern-matching) characters in these contexts:

- Attache commands — Attache expands wildcards in command pathnames with respect to the view, except in three cases—**import**, **lslocal**, and **put**—in which wildcards are expanded with respect to the workspace. In addition, various commands accept pattern arguments that can include wildcards; for example, see **find –name**, and **lsvob** *vob-tag-pattern*. In one of these arguments, a wildcard pattern must be quoted to protect it from evaluation by the command processor itself. For example:

  cmd-context **lsvob –region "dev*" "*src*"** *("pattern" arg; quotes required)*
  cmd-context **ls *.c** *(standard pname arg; no quotes required)*

- Config spec rules — The pathname pattern in a config spec rule is interpreted by a view's associated **view_server** process.

Attache recognizes these wildcard characters:

?	Matches any single character.
*	Matches zero or more characters.
~	Indicates your home directory on the helper host, except for **put**, **import**, and **lslocal** commands.
~*username*	Indicates *username*'s home directory on the helper host (except for **put**, **import**, and **lslocal** commands).
[*xyz***]**	Matches any of the listed characters.
[*x-y*]	Matches any character whose ASCII code falls between that of *x* and that of *y*, inclusive.

`...` (Ellipsis, a ClearCase extension) Matches zero or more directory levels.

For example:

foo/.../bar matches any of the following pathnames:

```
foo/bar
foo/usr/src/bar
foo/rel3/sgi/irix5/bar
```

and

foo/... matches the **foo** directory itself, along with the entire directory tree under it.

See the **config_spec** reference page for more information, including restrictions.

wildcards_ccase

Pattern-matching characters for ClearCase and ClearCase LT pathnames

APPLICABILITY

Product	Command Type
ClearCase	general information
ClearCase LT	general information

NOTE: For Attache, see the **wildcards** reference page.

SYNOPSIS

? ***** **~** **[...]** **...**

DESCRIPTION

Wildcard (pattern-matching) characters are recognized in these contexts:

- **cleartool** single-command mode—The operating system shell, not **cleartool**, interprets pathnames and expands wildcards. With some **cleartool** commands (**catcr –select**, **find –name**, **lsvob**), you can specify a pathname pattern as a quoted argument; these are always interpreted by **cleartool**:

  cleartool catcr –select "bug?.o" bgrs@@04-Mar.22:54.426

- **cleartool** interactive mode—**cleartool** expands wildcards in pathnames. In **cleartool** commands that accept *pattern* arguments (**catcr –select**, **find –name**, and **lsvob**), you must quote a wildcard pattern to protect it from evaluation by **cleartool** itself. For example:

  ```
  cleartool> lsvob –region "dev*"  "*src*"      ("pattern" arg; quotes required)
  cleartool> ls *.c                             (standard pname arg; no quotes required)
  ```

- Config spec rules — The pathname pattern in a config spec rule is interpreted by a view's associated **view_server** process.

ClearCase and ClearCase LT recognize these wildcard characters:

?	Matches any single character.
*	Matches zero or more characters.
~	Indicates your home directory.
[*xyz*]	Matches any of the listed characters.
[*x-y*]	Matches any character whose ASCII code falls between that of *x* and that of *y*, inclusive.

`...` Ellipsis; matches zero or more directory levels.

For example:

foo/.../bar matches any of the following pathnames:

```
foo/bar
foo/usr/src/bar
foo/rel3/sgi/irix5/bar
```

and:

foo/... matches the **foo** directory itself, along with the entire directory tree under it.

See the **config_spec** reference page for more information, including restrictions.

winkin

Accesses one or more *derived objects* (DOs) from a *dynamic view*, or converts a nonshareable derived object to a shareable (promoted) derived object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
Attache	command

SYNOPSIS

- Wink in a single DO or a list of explicitly named DOs:

  winkin [**–pri**·**nt**] [**–nov**·**erwrite**] [**–sib**·**lings** [**–adi**·**rs**]]
  [**–out** *pname*] *do-pname ...*

- Recursively wink in a DO and all of its subtargets:

  winkin [**–pri**·**nt**] [**–nov**·**erwrite**] [**–r**·**ecurse** [**–adi**·**rs**] [**–sel**·**ect** *do-leaf-pattern*] [**–ci**]]
  do-pname ...

DESCRIPTION

The **winkin** command enables you to access the data of any existing DO, even if it does not match your view's build configuration (and, thus, would not be winked in by a **clearmake** build). Note that you cannot access a DO's file-system data directly, using a *version-extended pathname*, such as **hello@@21-Dec.16:18.397**. Instead, you must wink it in to a dynamic view, and then access it using that view.

winkin also converts nonshareable DOs to shareable (promoted) DOs. If you specify a nonshareable DO, **winkin** first advertises the DO by writing information about it to the VOB, and then promotes it by copying its data container into the VOB and moving its configuration record into the VOB. Because a shareable DO cannot have nonshareable sub-DOs or sibling DOs, winking in a nonshareable DO also advertises its sub-DOs and siblings, converting them to shareable DOs. With **–siblings**, **winkin** advertises and promotes the DO's siblings.

NOTE: When a nonshareable DO is converted to a shareable DO, its DO-ID changes. For more information, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*.

Effect on View-Resident DO Data Containers

If you specify a shared DO while working in the view where it was originally built, and if a view-resident data container for the DO in that view still exists, then the view-resident data

container is scrubbed, and your view accesses the shared data container in VOB storage. This is equivalent to executing a **view_scrubber** command.

If you specify an unshared DO or nonshareable DO in your view, the data container is promoted to the VOB. The view-resident data container is scrubbed, and your view accesses the data container in VOB storage. This is equivalent to executing a **view_scrubber –p** command.

When you need to process a large number of DOs, use **view_scrubber** rather than **winkin**.

PERMISSIONS AND LOCKS

No special permissions are required at the VOB database level. No locks apply. At the file-system level, you must have read permission on the DO to be winked in. If you are overwriting an existing DO in your view (perhaps one that was winked in previously), you must have write permission on the existing DO. See the **CCASE_BLD_UMASK** discussion in the **clearmake** reference page.

OPTIONS AND ARGUMENTS

LISTING RESULTS INSTEAD OF PERFORMING THE WINKIN. *Default:* The listed derived objects are winked in.

–pri·nt
: Lists the names of DOs that would be winked in without winking them in. This option is useful for previewing what will happen before committing to a winkin operation that could overwrite a large number of derived objects in the view.

PRESERVING UNSHARED DERIVED OBJECTS IN YOUR VIEW. *Default:* **winkin** overwrites any unshared DOs in your view.

–nov·erwrite
: Preserves the unshared DOs in your view. Unshared DOs are often a result of checked-out source files. This option is useful to help limit winkins over DOs that were created from those source files.

WINKING IN SIBLING DERIVED OBJECTS. *Default:* Only the listed DOs are winked in, without their siblings (DOs created by the same build script that created the DO to be winked in). Note that you do not need to use **–siblings** with **–recurse**, which always winks in siblings.

–sib·lings
: Winks in the siblings of this derived object in addition to the derived object itself.

WINKING IN DERIVED OBJECT SUBTARGETS. *Default:* Only the listed derived objects are winked in, without any derived objects that are subtargets of these objects. Only derived objects in directories rooted at the current working directory are winked in.

–r·ecurse
: Recursively winks in all subtargets of the listed derived objects (subject to the

restrictions specified by other options). This option works by recursively walking the configuration records containing those DOs, gathering information about which subtargets to wink in, and weeding out duplicates. The gathered names are then winked in.

If multiple versions of the same object appear in a derived object's configuration, only the most recent version is winked in. A warning tells you which version is being skipped.

winkin –recurse keeps going even if the winkin of one or more of the items in the configuration record hierarchy fails, though the command issues errors for the ones that failed.

Because this command winks in derived objects without regard to any makefile information, it is usually a good idea to run **clearmake** after performing this operation, to bring everything up to date.

–adi·rs

Allows winkin to directories other than those rooted at the current directory.

–adirs only has effect with **–recurse** or **–siblings**.

–sel·ect *do-leaf-pattern*

(For use in recursive winkins only) Starts gathering the list of files to wink in at the subtargets of *do-pname* that match the specified pattern. *do-leaf-pattern* can be a pattern (see the ClearCase **wildcards_ccase** or Attache **wildcards** reference pages) that matches a simple filename; it must not include a slash (*/*) or the ellipsis wildcard (**...**). Alternatively, it can be a standard pathname of a derived object.

This option is useful for isolating a derived object that was built as a dependency of another one. For example, this command winks in derived objects starting at the **hello.o** that was used to build **hello** in the current view:

cmd-context **winkin –recurse –select hello.o hello**

–select only has effect with **–recurse**.

–ci

(For use in recursive winkins only) By default, recursive winkins stop at DO versions: DOs that have been checked in as versions of elements, and used as sources during the build. This option allows you to recurse into the CRs of DO versions. **–ci** only has effect with **–recurse**.

SPECIFYING AN ALTERNATIVE PATHNAME. *Default:* A derived object is winked in to your view at the pathname you specify with a *DO-pname* argument, minus any DO-ID. For example, if you specify the DO-pname **../src/hello@@21-Dec.16:18.397**, then by default, it is winked in at pathname **../src/hello**. Any object at the destination pathname is overwritten, subject to standard

permissions-checking. (Overwriting a shared DO decrements its reference count; no file system data is actually deleted.)

–out *pname*

An alternative pathname at which to wink in the DO. You must specify exactly one DO in this case, and *pname* must be in the same VOB as the DO being winked in.

- If *pname* is a directory, the DO is winked in to that directory, with the same leaf name as the original DO.
- Otherwise, *pname* is treated as a file name.

In either case, an error occurs if an object already exists at the destination. **–out** only has effect without the **–recurse** option.

NOTE: You must use **–out** if you are not using **–recurse** and specify another view's DO, using a view-extended pathname, and you intend to wink in the DO to your own view.

SPECIFYING THE DERIVED OBJECT. *Default:* None.

do-pname ...

One or more pathnames that specify derived objects. A standard pathname names a DO in the current view; you can also use a view-extended pathname and/or a VOB-extended pathname:

`/view/george/users_hw/hello`	*(view-extended pathname)*
`hello@@21-Dec.16:18.397`	*(VOB-extended pathname, including DO-ID)*
`/view/george/users_hw/hello@@05-Jan.09:16:788`	*(combination)*

EXAMPLES

These examples are written for use in **csh**. If you use another shell, you may need to use different quoting and escaping conventions.

In **cleartool** single-command mode, *cmd-context* represents the shell prompt. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Wink in another view's DO into your view, using a view-extended pathname. The **–out** option is required in this case.

  cmd-context **winkin –out . /view/george/usr/hw/hello.o**
  ```
  Winked in derived object "hello.o"
  ```

- Wink in a DO, using its DO-ID, and saving it under another file name.

  cmd-context **lsdo hello**
  ```
  02-Mar.20:02 "hello@@02-Mar.20:02.376"
  01-Mar.09:06 "hello@@01-Mar.09:06.365"
  ```

cmd-context **winkin –out hello.March1 hello@@01-Mar.09:06.365**
```
Promoting unshared derived object "hello@@01-Mar.09:06.365"
Winked in derived object "hello.March1"
```

- Create a new derived object and promote it to VOB storage.

  clearmake

  ```
  cc -c hello.c
  cc -c util.c
  cc -o hello hello.o util.o
  ```

  cmd-context **winkin hello**
  ```
  Promoting unshared derived object "hello"
  Winked in derived object "hello"
  ```

- Wink in derived object **main.o** and all of its siblings.

  cmd-context **lsdo main.o**
  ```
  04-Sep.16:14   "main.o@@04-Sep.16:14.49"
  ```

  cmd-context **winkin –siblings main.o@@04-Sep.16:14.49**
  ```
  Promoting unshared derived object "/vobs/mg_test/main.o".
  Winked in derived object "/vobs/mg_test/main.o"
  Promoting unshared derived object "/vobs/mg_test/sibling".
  Winked in derived object "/vobs/mg_test/sibling"
  ```

- Recursively wink in derived object **main** and all of its subtargets.

  cmd-context **winkin –recurse main@@04-Sep.16:03.34**
  ```
  Promoting unshared derived object "/vobs/mg_test/main"
  Winked in derived object "/vobs/mg_test/main
  Promoting unshared derived object "/vobs/mg_test/main.o"
  Winked in derived object "/vobs/mg_test/main.o"
  Promoting unshared derived object "/vobs/mg_test/sibling"
  Winked in derived object "/vobs/mg_test/sibling"
  Promoting unshared derived object "/vobs/mg_test/test.o"
  Winked in derived object "/vobs/mg_test/test.o"
  ```

  NOTE: When you use **–recurse**, you can also specify the DO to wink in by using its view-extended pathname. The DO and its subtargets are recursively winked in to the current (dynamic) view. For example:

  cmd-context **winkin –recurse /view/cep/vobs/mg_test/main**

- List the DOs that would be winked in during a recursive winkin of derived object **main**.

  cmd-context **winkin –print –recurse main@@04-Sep.16:03.34**
  ```
  Would wink in derived object "/vobs/mg_test/main"
  Would wink in derived object "/vobs/mg_test/main.o"
  Would wink in derived object "/vobs/mg_test/test.o"
  ```

- Recursively wink in derived object **main** and all of its subtargets, preserving the unshared DOs in your view.

  cmd-context **winkin –noverwrite –recurse /view/test/vobs/mg_test/main**

```
Winked in derived object "/vobs/mg_test/main"
Winked in derived object "/vobs/mg_test/main.o"
Winked in derived object "/vobs/mg_test/sibling"
Will not wink in over unshared derived object "/vobs/mg_test/test.o"
```

SEE ALSO

clearmake, **scrubber**, **view_scrubber**

ws_helper

Server process connecting an Attache workspace to a ClearCase view

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

Invoked on the helper host by the **albd_server** as a result of the **mkws** or **setws** commands

DESCRIPTION

The *workspace helper* program is the process managing the connection between the Attache workspace and ClearCase views and VOBs. There is a one-to-one mapping between Attache workspace users and workspace helper processes. Executing a **mkws** or **setws** command starts **ws_helper** on the *helper host*, which remains active until you exit the workspace. The **–shost** option of the **mkws** command specifies the host on which **ws_helper** runs. By default this is the view host, if the view can be found in the ClearCase registry.

ws_helper provides the following basic services:

- ClearCase command execution
- File transfer between the workspace and the view
- Wildcard lookup
- Other file-system support

ws_helper is invoked by the **albd_server** and inherits its environment. Then **ws_helper** assumes the identity of the user whose identity you provide; however, it does not acquire the user's environment automatically.

On UNIX hosts, **ws_helper** sets its **umask** to zero, adds the directory **ccase-home-dir/bin** to the **PATH** environment variable, and sets the **HOME**, **LOGNAME**, **USER**, and **SHELL** environment variables after a user has been successfully authorized. On Windows NT hosts, **ws_helper** adds the directory *ccase-home-dir***bin** to the **PATH** environment variable, and sets the **HOME** environment variable (if the user's domain account specifies a home directory and the **HOME** environment variable is not already set as a system variable).

CONFIGURING THE HELPER ENVIRONMENT

You can create a program or script in the **bin** subdirectory of **ccase-home-dir** on the *helper host* to set up and configure the environment in which the Attache helper is invoked. On a UNIX host, the program or script must be named **ws_startup**; on a Windows NT host, the program or script

must be named **ws_startup.***ext*, where *ext* is **bat**, **com**, **exe,** or any other extension that **cmd.exe** recognizes to be executable. This program or script can be used to perform security checks, config spec validation, or to set the environment variables needed by any program **ws_startup** invokes.

NOTE: If **ws_startup** is a UNIX script, it must be executable for all Attache users, and the first line must be: **#! shell**, where **shell** is the path name to the appropriate shell, for example, **/bin/csh**.

ws_startup can also make use of the environment variables set before it is run, namely:

- **ATTACHE_USER**, the authorized user's user name
- **ATTACHE_VIEW_TAG**, the view tag (or workspace name) to which the helper has been set
- **ATTACHE_ENV_PN**, the name of the temporary file in which a user's environment variables can be set

To set user-specific environment variables, for example, those needed to run *triggers*, you must have a properly configured file on the *helper host*:

- If there is a predefined set of environment variables, you can create a file named **.attache.env** in the user's home directory.

  On Windows NT, the helper determines where the user's home directory is by going to the domain controller, which validates the logon and checks the user's profile for a directory path under "Home Directory". For the helper to get to the user's home directory, the directory must be a UNC path because the user's home directory is not necessarily local to the helper host.

- If the information is dynamic, you can have **ws_startup** create the temporary file whose name can be read from **ATTACHE_ENV_PN**, and write the environment variables to it.

The workspace helper program attemptsto read environment settings first from the file named in **ATTACHE_ENV_PN** and, only if that file does not exist, from **.attache.env**.

Entries in the file must appear one per line. For each environment variable **ENV_NAME** you want to set to the value *val*, add an entry of the form **ENV_NAME**=*val*. For each environment variable you want to unset, add an entry of the form **ENV_NAME**=.

For UNIX helper hosts, you can also control the **umask** of the helper process by adding an entry of the form **umask**=*val*, where *val* must be expressed in C integer constant format. That is, *val* can be a decimal number beginning with the integers 1–9, an octal number beginning with zero, or a hexadecimal number beginning with `0x` or `0X` (a zero followed by an "x"). Instead of setting an environment variable, the helper sets its own **umask** to *val*. On Windows NT helpers, this entry type is ignored.

PERMISSIONS

Certain permissions are required to use the **ws_helper** program on Windows NT:

- The identity under which the **albd_server** (Atria Location Broker) runs must have local *Administrator* privileges. This is necessary so that the helper program can create services with the **Service Control Manager**.
- Any authorization identity given to Attache by a client must have "logon as a service" privileges on the helper host.

See the *ClearCase Attache Installation and Release Notes* for details.

ERROR LOG

The **ws_helper** sends warning and error messages to the event log on Windows NT hosts, or to **/var/adm/atria/log/ws_helper_log** on UNIX hosts.

SEE ALSO

mkws, **setws**, **albd_server**, **attache**

wshell

Executes a local shell in the current working directory of the workspace

APPLICABILITY

Product	Command Type
Attache	command

SYNOPSIS

wsh·ell [*command* [*arg* ...]]

DESCRIPTION

Thw **wshell** command executes a local shell in the current working directory; this directory must exist locally. An initial command can be specified. The shell runs interactively until you exit from it. On Windows 3.x, the **wshell** command invokes *attache-home-dir***etc\wshell.pif** which can be customized to run the shell of your choice. On Windows NT and Windows 95, **wshell** looks for an environment variable named **COMSPEC**. If found, the value of the **COMSPEC** environment variable is used as the name of the shell program to run. Otherwise, **wshell** runs **cmd.exe**, which must be found on your **PATH**.

PERMISSIONS AND LOCKS

Permissions Checking: No special permissions required. *Locks:* No locks apply.

OPTIONS AND ARGUMENTS

Initial Command. *Default:* **wshell** runs no initial command by default.

command [*arg* ...]
: The interactive shell invokes the program *command*, (and, optionally, passes it one or more arguments). The shell remains after the command is executed.

EXAMPLES

- Create a new window running an interactive shell.

```
/vob/src> pwd
/vob/src
/vob/src> wshell
```

- Create a new window running an interactive shell that runs a **dir** command.

```
/vob/src> pwd
/vob/src

/vob/src> wshell dir *.c
```

SEE ALSO

shell, make, attache_command_line_interface

xclearcase

Primary ClearCase and ClearCase LT graphical interface utility

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

- Start in File Browser:
  xclearcase [*X-options*] [**–file**] [**–ngraph**] *pname ...*
- Start in Version Tree Browser:
  xclearcase [*X-options*] **–vtree** [**–all**] [**–nme·rge**] [**–nco**]
  [**–lab·el**] [**–ngraph**] [**–tag** *view-tag*[**,**...]] *pname ...*
- Start in Hyperlink Tree Browser:
  xclearcase [*X-options*] **–htr·ee** [**–dir·ect** | { **–nel·ement** | **–nbr·anch** | **–nve·rsion** }] ...
  [**–inc·lude** *hlink-type*[**,**...] | **–exc·lude** *hlink-type*[**,**...]] [**–to_only** | **–fro·m_only**]
  [**–tex·t**] [**–ngraph**] *pname ...*
- Start inType Browser:
  xclearcase [*X-options*] { **–att·ype** | **–brt·ype** | **–elt·ype** | **–hlt·ype** | **–lbt·ype** | **–trt·ype** }

DESCRIPTION

The **xclearcase** command invokes the ClearCase or ClearCase LT GUI (graphical user interface).

xclearcase is implemented as an X Window System application using a standard window system toolkit. See your X Window System documentation for a description of mouse and keyboard conventions.

OPTIONS AND ARGUMENTS

SELECTING A BROWSER. *Default:* Starting **xclearcase** brings up the **main panel**, an enhanced file browser. (**xclearcase –file** has the same effect.)

–file [**–ngraph**] *dir-pname ...*
Starts a separate file browser on each of the specified VOB directories (or a single browser on the current working directory). Using **–ngraph** starts non-graphical browsers; objects are listed by name instead of being displayed as icons.

–att·ype , **–brt·ype** , **–elt·ype** , **–hlt·ype** , **–lbt·ype** , **–trt·ype**
Starts a single type browser.

–vtr·ee [**–all**] [**–nme·rge**] [**–nco**] [**–tag** *view-tag+*] *pname ...*
Similar to the **cleartool lsvtree –graphical** command. See the **lsvtree** reference page for details on the **–all**, **–nmerge**, and **–nco** options.

–htr·ee [*options*]
Starts a Hyperlink Tree Browser. By default, the browser starts in inheritance mode; use **–direct** to start it in direct mode.

By default, there is no filtering of hyperlinks by kind of file-system object; use **–nelement**, **–nbranch**, and/or **–nversion** to exclude links whose left ends are connected to certain kinds of objects.

By default, there is no filtering of hyperlinks by direction; use **–from_only** or **–to_only** to restrict the display to hyperlinks from/to the specified *pname*.

By default, hyperlinks of all types except **Merge** are displayed; use **–include** or **–exclude** to specify exactly which types to display.

By default, annotations to hyperlinks are indicated by a single quote ('); use **–text** to display full annotations.

By default, the browser uses a graphical display (icons); use **–ngraph** to have the browser start in text mode.

X WINDOW SYSTEM OPTIONS. *Default:* None.

X-options
xclearcase accepts all the standard X Toolkit command-line options (for example, **–display**), as described in the **X(1)** reference page. Quote the option string if it includes white space.

X RESOURCES

Following are the shell instance names for the **xclearcase** browsers and transcript pad:

```
xclearcase.vtree
xclearcase.metatype
xclearcase.file
xclearcase.viewtag
xclearcase.vob
xclearcase.username
xclearcase.string
xclearcase.list
xclearcase.pool
xclearcase*transcript
```

The **vtreeTagColors** resource takes a comma-separated list of colors. For example:

```
xclearcase.vtreeTagColors: DarkOliveGreen4,Blue,IndianRed2
```

The colors on this list are used for the view-tag annotations specified by the **–tag** option.

You can use the **useSmallFonts** resource to increase the density of the File Browser's display, both when displaying file names and when displaying icons:

```
xclearcase.useSmallFonts: True
```

EXAMPLES

- Start the graphical user interface.

  % **xclearcase**

- Start the graphical user interface in view **test_vu**.

  % **cleartool setview –exec xclearcase test_vu**

- Display the version tree for element **base.c**, annotating the version selected by view **dvt_vu** in blue, and the version selected by view **fix_vu** in red.

  % **xclearcase -xrm "xclearcase.vtreeTagColors: blue,red" -vtree -tag dvt_vu,fix_vu base.c**

SEE ALSO

X(1)

xcleardiff

Compares or merges text files graphically

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

SYNOPSIS

- Compare files:
  **xcleardiff [–b·lank_ignore] [–tin·y] [–html] [–hst·ack | –vst·ack] [*X-options*] *pname1 pname2 ...*
- Merge files:
  xcleardiff –out *output-pname* [**–bas·e** *pname*] [**–tin·y**] [**–html**]
  [**–hst·ack** | **–vst·ack**]
  [**–q·uery** | **–qal·l** | **–abo·rt**] [*X-options*] *contrib-pname ...*

DESCRIPTION

xcleardiff is a graphical diff and merge utility for text files. It implements the **xcompare** and **xmerge** methods for the **text_file** and **compressed_text_file** type managers, as well as the graphical portions of these methods for the **directory** and **_html** type managers. On color display monitors, **xcleardiff** uses different colors to highlight changes, insertions, and deletions from one or more contributing files. During merge operations, contributors are processed incrementally and, when necessary, interactively, to visibly construct a merge results file. You can edit this ts file directly in the merge results pane as it is being built to add, delete, or change code manually, or to add comments.

xcleardiff is implemented as an X Window System application using a standard Motif toolkit. See your X Window System documentation for a description of general mouse and keyboard conventions.

INVOKING xcleardiff

You can invoke **xcleardiff** directly from the command line, specifying files or versions to compare or merge. Invoking **xcleardiff** directly bypasses the type managers, so invoke **xcleardiff** directly only when you are working with text files that are not stored in a VOB.

The following **cleartool** subcommands, when applied to text files, also invoke **xcleardiff**:

- **diff –graphical**

- **merge –graphical**
- **findmerge** (with options **–graphical** or **–okgraphical**)

The **findmerge** command includes the advantage of some extra command options—optional preprocessing—in the same way that **diff** and **merge** offer more flexibility than direct calls to the character-based **cleardiff** utility. See **findmerge –ftag**, for example.

Various buttons and commands in the **xclearcase** graphical interface also invoke **xcleardiff**.

NOTE: When comparing/merging HTML files, if the machine on which you execute **xcleardiff** is not the machine on which you run your HTML browser, your browser may not be able to find the pathname to the files being compared/merged.

CHANGING THE DEFAULT HTML BROWSER

xcleardiff invokes a script to determine which HTML browser to use when comparing or merging files of type **html**. By default, **xcleardiff** invokes the Netscape browser through the script **display_url.sh**. To change the default values, use the following environment variables:

CCASE_WEB_SCRIPT (Default value: **$ATRIAHOME/etc/display_url.sh**)
Invokes the script specified, which designates the browser to use.

CCASE_NETSCAPE (Default value: "netscape")
Changes the default version of the Netscape browser to the specified version. If the Netscape browser is accessible through **$PATH**, you need only specify the executable name; if it is not in your path specification, you must specify a full pathname.

CCASE_NETSCAPE_OPT (Default value: `NULL`)
Provides additional command-line options to the Netscape browser through the script, for example, **–install** to force the Netscape browser to use a private colormap.

OPTIONS AND ARGUMENTS

HANDLING OF WHITE SPACE. *Default:* When comparing files, **xcleardiff** pays attention to changes in white space.

–blank_ignore
(Valid only when comparing files, not when merging) Causes **xcleardiff** to ignore extra white space characters in text lines: leading and trailing white space is ignored; internal runs of white-space characters are treated like a single **<SPACE>** character.

FONT SIZE. *Default:* **xcleardiff** uses the font specified by the resource `xcleardiff*diffFontSet`.

–tiny
Uses a smaller font, to increase the amount of text displayed in each pane.

INVOKING THE TYPE MANAGER FOR HTML FILES. *Default:* When **xcleardiff** is invoked directly, the type manager is bypassed. When **xcleardiff** is invoked indirectly (through **cleartool** or the graphical interface), the type manager is used.

–html
Starts the **_html** type manager.

CONTRIBUTOR PANE STACKING. *Default:* Each of the two or more files being compared or merged is displayed in a separate subwindow, or contributor pane. By default, these panes are displayed, or stacked, horizontally (side by side), with the base contributor on the left.

–vst·ack
Stacks the contributor panes vertically, with the base contributor at the top.

–hst·ack
Displays the contributor panes horizontally (the default behavior).

MERGE RESULTS FILE. *Default:* None. You must specify a merge results file with the **–out** option.

–out *output-pname*
(Merge only; required) Specifies the merge results file, either a checked-out version or a standard operating-system file.

SPECIFYING A BASE CONTRIBUTOR FOR A MERGE OPERATION. *Default:* **xcleardiff** does not calculate a base contributor (see the **merge** reference page). The first contributor named on the command line becomes the base contributor, against which the one or more additional contributors are compared. Query on All mode (**–qall**) is in effect by default, but can be deactivated from the graphical interface.

–bas·e *pname*
(merge only) Makes *pname* the base contributor for a merge. Using **–base** turns off "Query on All" mode, unless **–qall** is explicitly supplied. See also *Merge Automation*.

MERGE AUTOMATION. *Default:* If you do not specify a base contributor with **–base**, Query on All mode is enabled. In this mode, **xcleardiff** prompts you to accept or reject each change, insertion, or deletion found in contributors 2 through *n* on the command line. The options described in this subsection have no effect.

If you specify a base contributor with **–base**, **xcleardiff** performs the merge automatically, prompting only if two or more contributors modify the same section of the base contributor. If all changes can be merged automatically, **xcleardiff** prompts you before saving the merge results file.

–q·uery
–qal·l
–abo·rt (mutually exclusive)
–query turns off automatic merging for nontrivial merges (where two or more

contributors differ from the base file) and prompts you to proceed with every change in the from-versions. Changes in the to-version are accepted unless a conflict exists.

–qall turns off automatic acceptance of changes in which only one contributor differs from the base file. In this mode, **xcleardiff** prompts you to accept or reject each modification (relative to the base file) in each contributor, as it does when two or more contributors differ from the base contributor. You can switch this mode interactively during the **xcleardiff** session.

–abort is intended for use with scripts or batch jobs that involve merges. It allows completely automatic merges to proceed, but aborts any merge that would require user interaction.

X WINDOW SYSTEM OPTIONS. *Default:* None.

X-options

xcleardiff accepts all the standard X Toolkit command-line options (for example, **–display**), as described in the **X(1)** reference page. If the option string includes white space, enclose it in quotes.

DIFF/MERGE CONTRIBUTORS. *Default:* None. You must specify at least two files for a **diff** operation, and at least one file for a merge operation (two, if a base contributor is not supplied with **–base**).

contrib-pname ...

The files to be compared or merged. If a merge operation does not explicitly include a base contributor with **–base**, the first *contrib-pname* becomes the base contributor. For a diff operation, **xcleardiff** does not calculate a common ancestor (see the **diff** reference page); the first *contrib-pname* is the base contributor against which subsequent contributors are compared.

EXAMPLES

- Compare two files in different directories.

  % **xcleardiff test.c ~jpb/my_proj/test_NEW.c**

- Compare two HTML files and invoke the type manager for HTML files.

  % **xcleardiff –html my_new_source.html my_old_source.html**

SEE ALSO

cleardiff, **diff**, **merge**, **schemes**, **type_manager**

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z