ClearCase Product Family Documentation Supplement

Release 4.2

Windows/UNIX Edition

Rational®
the e-development company™

800-024446-000 (patch)

ClearCase Product Family Documentation Supplement
Document Number 800-024446-000 (patch) June 2001
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 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

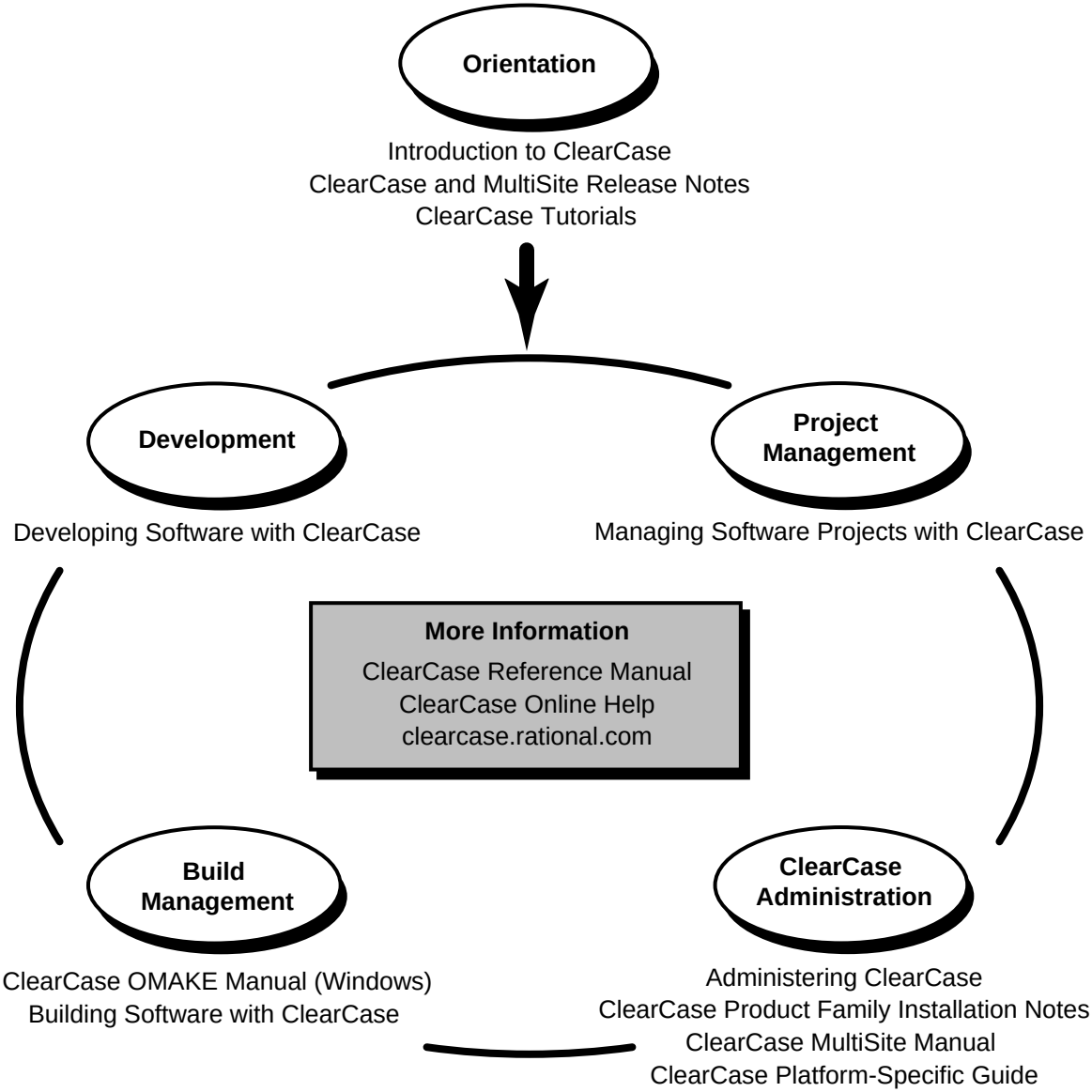
Preface

This manual contains documentation about new features in ClearCase and ClearCase LT for Release 4.2. Some of the information supersedes the chapters and reference pages in existing ClearCase and ClearCase LT manuals.

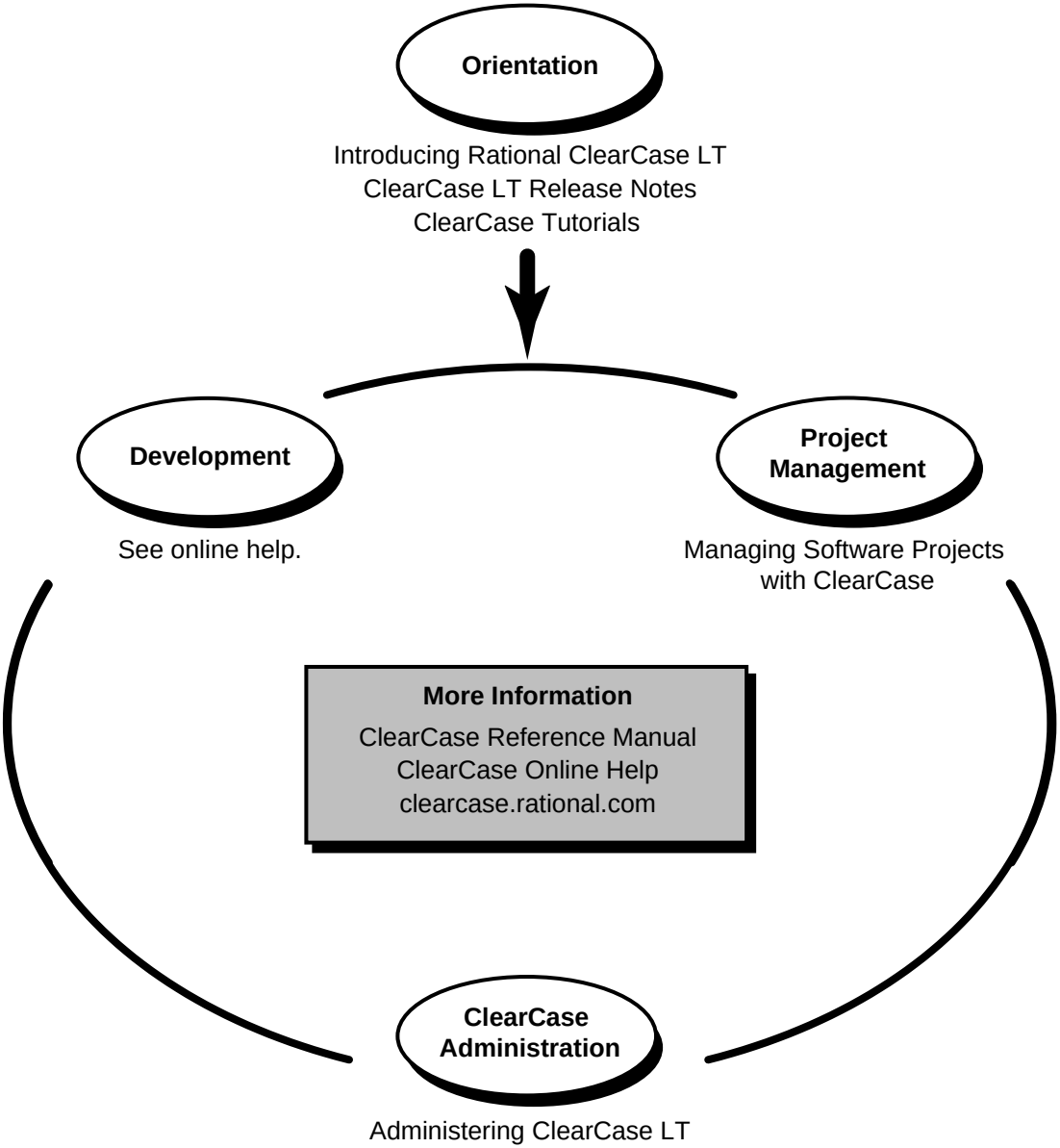
About This Manual

This manual is intended for all ClearCase, ClearCase LT, and MultiSite users.

ClearCase Documentation Roadmap



ClearCase LT 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).

Online Documentation

The ClearCase Product Family (CPF) graphical interfaces include an online 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)*
multitool man mkreplica	*(display the multitool mkreplica 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 uncheckout –help
```
Usage: uncheckout | unco [-keep | -rm] [-cact | -cwork ] pname ...
```

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

On UNIX, 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**

1 Introduction

This chapter describes the material included in this manual. The chapters and reference pages in this manual supersede the information in the Release 4.1 manuals.

1.1 UCM Information

New in this release is UCM trigger support. Chapter 2, *Using Triggers to Enforce Development Policies* (from the *Managing Software Projects with ClearCase* manual) and the **mktrigger**, **mktrtype**, and **rmtrigger** reference pages contain information about UCM triggers.

Chapter 3, *Setting Up a ClearQuest User Database*, contains new information about using MultiSite to replicate the PVOB or ClearQuest user database involved in the UCM-ClearQuest integration.

The *Restrictions* sections of all UCM reference pages have been updated with new information on locks and required identities. The operations listed below require a privileged identity (for all other UCM operations, no special identity is required):

- **chproject**
- **rmactivity**
- **rmbl**
- **rmcomp**
- **rmfolder**
- **rmproject**
- **rmstream**

For specific information, see the UCM reference pages.

1.2 Administrative Information

Chapter 4, *Cross-Platform File Access*, contains new information about configuring TAS 6.0 and Samba.

1.3 Build Information

The ClearCase build tools, **clearmake** and **omake**, have a new option to check out derived objects automatically before using them. The **clearmake** and **omake** reference pages contain information about this new option. These reference pages do not apply to ClearCase LT.

1.4 New Command to Import File-System Objects

The new **clearfsimport** command reads file-system source objects and places them in the target VOB. This command can be run in UCM views. The **clearfsimport** reference page describes this command.

1.5 Snapshot View Creation Option for mkview Command

In ClearCase Release 4.1, support for the **mkview –vws** option for snapshot view creation was withdrawn; Release 4.2 restores this functionality.

2 Using Triggers to Enforce Development Policies

UCM provides a group of development policies that you can easily set in a project by using the GUI or CLI. In addition, you can use triggers on certain UCM operations to enforce customized development policies for your project team. This chapter describes how to create triggers and shows how to use triggers to implement various development policies in UCM projects. For additional information, see the **cleartool mktrigger** and **mktrtype** reference pages.

2.1 Overview of Triggers

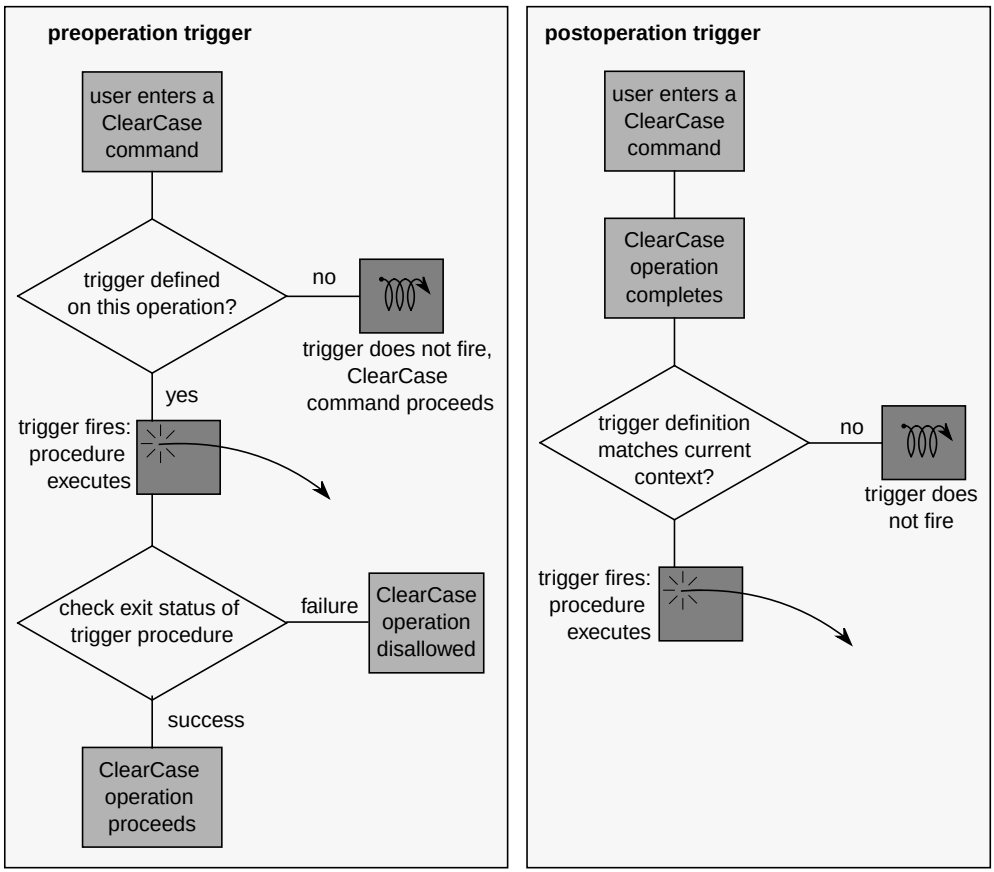
A *trigger* is a monitor that causes one or more procedures or actions to be executed whenever a certain ClearCase operation is performed. Typically, the trigger executes a Perl, batch, or shell script. You can use triggers to restrict operations to specific users and to specify the conditions under which they can perform those operations. You can use triggers with the following UCM operations:

- **deliver**
- **mkactivity**
- **mkbl**
- **mkstream**
- **rebase**
- **setactivity**

Preoperation and Postoperation Triggers

Triggers fall into one of two categories. Preoperation triggers *fire*, or execute their corresponding procedures, before an operation takes place. Postoperation triggers fire after an operation occurs. Use preoperation triggers to prevent users from performing operations unless certain conditions apply. Use postoperation triggers to perform actions after an operation completes. For example, you may want to place a postoperation trigger on the deliver operation to notify team members whenever a developer delivers work to the project's integration stream. Figure 1 illustrates the timing of preoperation and postoperation triggers.

Figure 1 Preoperation and Postoperation Triggers



Scope of Triggers

A *trigger type* defines a trigger for use within a VOB or PVOB. When you create a trigger type, with the **cleartool mktrtype** command, you specify the scope to be one of the following:

- An *element trigger type* applies to one or more elements. You attach an instance of the trigger type to one or more elements by using the **cleartool mktrigger** command.
- An *all-element trigger type* applies to all elements in a VOB.
- A *type trigger type* applies to type objects, such as attributes types, in a VOB.
- A *UCM trigger type* applies to a UCM object, such as a stream or a project, in a PVOB.
- An *all-UCM-object trigger type* applies to all UCM objects in a PVOB.

Using Attributes with Triggers

As you design triggers to enforce development policies, you may find it useful to use attributes. An *attribute* is a name/value pair. An *attribute type* defines an attribute. You can apply an attribute to an object, such as a stream or an activity, or to a version of an element. In your trigger scripts, you can test the value of an attribute to determine whether to fire the trigger. For example, you could define an attribute type called **TESTED** and attach a **TESTED** attribute to elements to indicate whether they had been tested. Acceptable values would be **Yes** and **No**.

When to Use ClearQuest Scripts Instead of UCM Triggers

This chapter presents several use cases for UCM triggers. If your UCM project is enabled to work with ClearQuest, you can set the following policies, which are described in *Managing Software Projects with ClearCase*:

- Check Before Work On
- Check Before ClearCase Delivery
- Do ClearQuest Action After Delivery
- Check Mastership Before Delivery

Each of these policies has a ClearQuest global hook script associated with it, which you can edit or replace in ClearQuest Designer to customize the policy for your environment. You can also

write your own ClearQuest hooks to enforce development policies. In general, if the policy you want to enforce involves a ClearQuest action, use one of the three ClearQuest policies listed above or use ClearQuest hooks. If the policy you want to enforce involves a ClearCase action, use UCM triggers.

2.2 Sharing Triggers Between UNIX and Windows

You can define triggers that fire correctly on both UNIX and Windows computers. The following sections describe two techniques. With one, you use different pathnames or different scripts; with the other, you use the same script for both platforms.

Using Different Pathnames or Different Scripts

To define a trigger that fires on UNIX, Windows, or both, and that uses different pathnames to point to the trigger scripts, use the **–execunix** and **–execwin** options with the **mktrtype** command. These options behave the same as **–exec** when fired on the appropriate platform (UNIX or Windows, respectively). On the other platform, they do nothing. This technique allows a single trigger type to use different paths for the same script or to use completely different scripts on UNIX and Windows computers. For example:

```
cleartool mktrtype –element –all –nc –preop checkin
–execunix /public/scripts/precheckin.sh –execwin \\neon\scripts\precheckin.bat
pre_ci_trig
```

NOTE: The command line example is broken across lines to make it easier to read. You must enter it all on one line.

On UNIX, only the script **precheckin.sh** runs. On Windows, only **precheckin.bat** runs.

To prevent users on a new platform from bypassing the trigger process, triggers that specify only **–execunix** always fail on Windows. Likewise, triggers that specify only **–execwin** fail on UNIX.

Using the Same Script

To use the same trigger script on Windows and UNIX platforms, you must use a batch command interpreter that runs on both operating systems. For this purpose, ClearCase includes the **ccperl** program, a version of Perl that you can use on Windows and UNIX.

The following **mktrtype** command creates sample trigger type **pre_ci_trig** and names **precheckin.pl** as the executable trigger script.

On UNIX:

```
cleartool mktrtype –element –all –nc –preop checkin   \
      –execunix 'Perl /public/scripts/precheckin.pl'   \
      –execwin 'ccperl \\neon\scripts\precheckin.pl'   \
      pre_ci_trig
```

On Windows:

```
cleartool mktrtype –element –all –nc –preop checkin ^
–execunix "Perl /public/scripts/precheckin.pl" ^
–execwin "ccperl \\neon\scripts\precheckin.pl" ^
pre_ci_trig
```

Tips

- To tailor script execution for each operating system, use environment variables in Perl scripts.
- To collect or display information interactively, you can use the **clearprompt** command.

2.3 Enforce Serial Deliver Operations

Because UCM allows multiple developers to deliver work to the same integration stream concurrently, conflicts can occur if two or more developers attempt to deliver changes to the same element. If one developer's deliver operation has an element checked out, the second developer cannot deliver changes to that element until the first deliver operation is completed or canceled. The second deliver operation attempts to check out all elements other than the checked-out one, but it does not proceed to the merge phase of the operation. The second

developer must either wait for the first deliver operation to finish, or undo the second deliver operation.

You may want to implement a development policy that eliminates the confusion that concurrent deliveries can cause developers. This section shows three Perl scripts that prevent multiple developers from delivering work to the same integration stream concurrently:

- Script 1 creates the trigger types and an attribute type.
- Script 2 is the preoperation trigger action that fires at the start of a deliver operation.
- Script 3 is the postoperation trigger action that fires at the end of a deliver operation.

Setup Script

This setup script creates a preoperation trigger type, a postoperation trigger type, and an attribute type. The preoperation trigger action fires when a deliver operation starts, as represented by the **deliver_start** operation kind (*opkind*). The postoperation trigger action fires when a deliver operation is canceled or completed, as represented by the **deliver_cancel** and **deliver_complete** opkinds, respectively.

The script runs on both UNIX and Windows platforms. Because the command-line syntax to run the preoperation and postoperation scripts on Windows differs slightly depending on whether the PVOB resides on Windows or UNIX, the setup script uses an **IF ELSE** Boolean expression to set the appropriate **execwin** command.

The **mktrtype** command uses the **–ucmobject** and **–all** options to specify that the trigger type applies to all UCM objects in the PVOB, but the **–stream** option restricts the scope to one integration stream.

The **mkattype** command creates an attribute type called **deliver_in_progress**, which the preoperation and postoperation scripts use to indicate whether a developer is delivering work to the integration stream.

```
# perl script to set up triggers for enforcing serial delivery.
use Config;

# define platform-dependent arguments.
my $PVOBTAG;
my $PREOPCMDW;
my $POSTOPCMDW;
if   ($Config{'osname'} eq 'MSWin32') {
     $PVOBTAG    = '\cyclone_pvob';
     $PREOPCMDW  = '-execwin "ccperl
\\\\pluto\disk1\ucmtrig_examples\ex1\ex1_preop.pl"';
     $POSTOPCMDW = '-execwin "ccperl
\\\\pluto\disk1\ucmtrig_examples\ex1\ex1_postop.pl"';
}
else {
     $PVOBTAG    = '/pvobs/cyclone_pvob';
     $PREOPCMDW  = '-execwin "ccperl
\\\\\\pluto\disk1\ucmtrig_examples\ex1\ex1_preop.pl"';
     $POSTOPCMDW = '-execwin "ccperl
\\\\\\pluto\disk1\ucmtrig_examples\ex1\ex1_postop.pl"';
}

my $PREOPCMDU    = '-execunix "Perl
/net/pluto/disk1/ucmtrig_examples/ex1/ex1_preop.pl"';
my $POSTOPCMDU   = '-execunix "Perl
/net/pluto/disk1/ucmtrig_examples/ex1/ex1_postop.pl"';
my $STREAM       = "stream:P1_int\@$PVOBTAG";
my $PREOPTRTYPE  = "trtype:ex1_preop\@$PVOBTAG";
my $POSTOPTRTYPE = "trtype:ex1_postop\@$PVOBTAG";
my $ATTYPE       = "attype:deliver_in_progress\@$PVOBTAG";

# set up the trigger types and attribute type.
print `cleartool mktrtype -ucmobject -all -preop deliver_start $PREOPCMDU
$PREOPCMDW -stream $STREAM -nc $PREOPTRTYPE`;
print `cleartool mktrtype -ucmobject -all -postop deliver_complete,
deliver_cancel $POSTOPCMDU $POSTOPCMDW -stream $STREAM -nc $POSTOPTRTYPE`;
print `cleartool mkattype -vtype integer -default 1 -nc $ATTYPE`;
```

Preoperation Trigger Script

This preoperation trigger action fires when a developer begins to deliver work to the specified integration stream. The script attempts to attach an attribute of type **deliver_in_progress** to the integration stream. If another developer is in the process of delivering work to the same stream, the **mkattr** command fails and the script displays a message suggesting that the developer try

again later. Otherwise, the **mkattr** command succeeds and prevents other developers from delivering to the integration stream until the current deliver operation finishes.

```
# perl script that fires on deliver_start preop trigger.
use Config;

# define platform-dependent arguments.
my $PVOBTAG;
if   ($Config{'osname'} eq 'MSWin32') {
     $PVOBTAG = '\cyclone_pvob';
}
else{
     $PVOBTAG = '/pvobs/cyclone_pvob';
}
my $STREAM = CLEARCASE_STREAM;
my $ATTYPE = "attype:deliver_in_progress\@$PVOBTAG";

# try to create the attribute, capture the output.
$msg = 'cleartool mkattr -default $ATTYPE $STREAM 2>&1';

# if the attribute already existed, a deliver is in progress, disallow this
delivery.
if   (index($msg, "Error: Object already has an attribute") >= 0) {
     print "***\n";
     print "*** A deliver operation is already in progress. Please try again
later.\n";
     print "***\n";
     exit 1;
}

# the attribute was created, deliveries will be disallowed until postop fires.
exit 0
```

Postoperation Trigger Script

This postoperation trigger action fires when a developer cancels or completes a deliver operation to the specified integration stream. This script removes the **deliver_in_progress** attribute that the preoperation script attaches to the integration stream at the start of the deliver operation. After the attribute is removed, another developer can deliver work to the integration stream.

```
# perl script that fires on deliver_complete or deliver_cancel postop trigger.
use Config;
```

```
# define platform-dependent arguments.
my $PVOBTAG;
if   ($Config{'osname'} eq 'MSWin32') {
     $PVOBTAG = '\cyclone_pvob';
}
else{
     $PVOBTAG = '/pvobs/cyclone_pvob';
}
my $STREAM = CLEARCASE_STREAM;
my $ATTYPE = "attype:deliver_in_progress\@$PVOBTAG";

# remove the attribute to allow deliveries.
print `cleartool rmattr -nc $ATTYPE $STREAM`;
exit 0;
```

2.4 Send Mail to Developers on Deliver Operations

To improve communication among developers on your project team, you may want to create a trigger type that sends an e-mail message to team members whenever a developer completes a deliver operation. This section includes two scripts:

- Script 1 creates a trigger type that fires at the end of a successful deliver operation.
- Script 2 is the postoperation trigger action that sends e-mail messages to developers.

Setup Script

This script creates a postoperation trigger type that fires when a developer finishes a deliver operation, as represented by the **deliver_complete** opkind. The **mktrtype** command uses the **–stream** option to indicate that the trigger type applies only to deliver operations that target the specified integration stream.

```
# This is a Perl script to set up the triggertype
# for e-mail notification on deliver.
use Config;
```

```
# define platform-dependent arguments.
my $PVOBTAG;
if   ($Config{'osname'} eq 'MSWin32') {
     $PVOBTAG  = '\cyclone_pvob';
     $WCMD     = '-execwin "ccperl
\\\\pluto\disk1\ucmtrig_examples\ex2\ex2_postop.pl"';
}
else {
     $PVOBTAG  = '/pvobs/cyclone_pvob';
     $WCMD     = '-execwin "ccperl
\\\\\\pluto\disk1\ucmtrig_examples\ex2\ex2_postop.pl"';
}
my $STREAM = "stream:P1_int\@$PVOBTAG";
my $TRTYPE = "trtype:ex2_postop\@$PVOBTAG";
my $UCMD   = '-execunix "Perl
/net/pluto/disk1/ucmtrig_examples/ex2/ex2_postop.pl"';

print 'cleartool mktrtype -ucmobject -all -postop deliver_complete $WCMD $UCMD
-stream $STREAM -nc $TRTYPE`;
```

Postoperation Trigger Script

This postoperation trigger action fires when a developer finishes delivering work to the integration stream. The script composes and sends an e-mail message to other developers on the project team telling them that a deliver operation has just finished. The script uses ClearCase environment variables to provide the following details about the deliver operation in the body of the message:

- Project name
- Development stream that delivered work
- Integration stream that received delivered work
- Integration activity created by the deliver operation
- Activities delivered
- Integration view used by deliver operation

```
# Perl script to send mail on deliver complete.

######################################################################
# Simple package to override the "open" method of Mail::Send so we
# can control the mailing mechanism.
```

```
package SendMail;

use Config;
use Mail::Send;

@ISA = qw(Mail::Send);

sub open {
     my $me = shift;
     my $how;  # How to send mail
     my $notused;
     my $mailhost;

     # On Windows use SMTP

     if ($Config{'osname'} eq 'MSWin32') {
     $how = 'smtp';
     $mailhost = "localmail0.company.com";
     }

     # else use defaults supplied by Mail::Mailer

     Mail::Mailer->new($how, $notused, $mailhost)->open($me);
}

#
################################################################################
# Main program

     my @to = "developers\@company.com";
     my $subject = "Delivery complete";

     my $body = join '', ("\n",
        "UCM Project: ", $ENV{CLEARCASE_PROJECT}, "\n",
        "UCM source stream: ", $ENV{CLEARCASE_SRC_STREAM}, "\n",
        "UCM destination stream: ", $ENV{CLEARCASE_STREAM}, "\n",
        "UCM integration activity: ", $ENV{CLEARCASE_ACTIVITY}, "\n",
        "UCM activities delivered: ", $ENV{CLEARCASE_DLVR_ACTS}, "\n",
        "UCM view: ", $ENV{CLEARCASE_VIEW_TAG}, "\n"
     );

     my $msg = new SendMail(Subject=>$subject);
```

```
    $msg->to(@to);
    my $fh  = $msg->open($me);
    $fh->print($body);
    $fh->close();
    1; # return success
#
#####################################################################
```

2.5 Do Not Allow Activities to Be Created on the Integration Stream

Anyone who has an integration view attached to the integration stream can create activities on that stream, but the UCM process calls for developers to create activities in their development streams. You may want to implement a policy that prevents developers from creating activities on the integration stream inadvertently. This section shows a Perl script that enforces that policy.

The following **mktrtype** command creates a preoperation trigger type called **block_integration_mkact**. The trigger type fires when a developer attempts to make an activity.

cleartool mktrtype -ucmobject -all -preop mkactivity -execwin "ccperl ^ \\pluto\disk1\triggers\block_integ_mkact.pl" -execunix "Perl ^ /net/jupiter/triggers/block_integ_mkact.pl" block_integration_mkact@\my_pvob

The following preoperation trigger script runs when the **block_integration_mkact** trigger fires. The script uses the **cleartool lsproject** command and the **CLEARCASE_PROJECT** environment variable to determine the name of the project's integration stream. ClearCase creates an integration activity to keep track of changes that occur during a deliver operation. The script uses the **CLEARCASE_POP_KIND** environment variable to determine whether the activity being created is an integration activity. If the **mkactivity** operation is the result of a deliver operation, the value of **CLEARCASE_POP_KIND**, which identifies the parent operation, is **deliver_start**.

If the value of **CLEARCASE_POP_KIND** is not **deliver_start**, the activity is not an integration activity, and the script disallows the **mkactivity** operation.

```
# Get the integration stream name for this project
my $istream = 'cleartool lsproject -fmt "%[istream]p"
$ENV{'CLEARCASE_PROJECT'}';

# Get the current stream and strip off VOB tag
$_ = $ENV{'CLEARCASE_STREAM'};
s/\@.*//;
my $curstream = $_;
```

```
# If it's the same as our stream, then it is the integration stream
if   ($istream eq $curstream) {
     # Only allow this mkact if it is a result of a deliver
     # Determine this by checking the parent op kind
     if ($ENV{'CLEARCASE_POP_KIND'} ne "deliver_start") {
        print "Activity creation is only permitted in integration streams for
        delivery.\n";
        exit 1
     }
}

exit 0
```

2.6 Implementing a Role-Based Access Control System

In a ClearCase environment, where users perform different roles, you may want to restrict access to certain ClearCase operations based on role. This section shows a trigger definition and script that implement a role-based access control system.

The following **mktrtype** command creates a preoperation trigger type called **role_restrictions**. The trigger type fires when a user attempts to make a baseline, stream, or activity.

**cleartool mktrtype -nc -ucmobject -all -preop mkstream,mkbl,mkactivity **
**-execunix "perl /net/jupiter/triggers/role_restrictions.pl" **
**-execwin "ccperl \\pluto\disk1\triggers\role_restrictions.pl" **
role_restrictions@\my_pvob

Preoperation Trigger Script

The following preoperation trigger script maps users to the following roles:

- Project manager
- Integrator
- Developer

The script maps the **mkactivity**, **mkbl**, and **mkstream** operations to the roles that are permitted to perform them. For example, only users designated as project managers or integrators can make a baseline.

The script uses the **CLEARCASE_USER** environment variable to retrieve the user's name, the **CLEARCASE_OP_KIND** environment variable to identify the operation the user attempts to perform, and the **CLEARCASE_POP_KIND** environment variable to identify the parent operation. If the parent operation is **deliver** or **rebase**, the script does not check permissions.

```
use strict;

sub has_permission
{
     my ($user,$op,$pop,$proj) = @_;

     #When performing a composite operation like 'deliver' or 'rebase',
     #we don't need to check permissions on the individual sub-operations
     #that make up the composite.

     return 1 if($pop eq 'deliver_start' || $pop eq 'rebase_start' ||
                 ($pop eq 'deliver_complete' || $pop eq 'rebase_complete' ||
                 ($pop eq 'deliver_cancel' || $pop eq 'rebase_cancel');

     # Which roles can perform what operations?
     # Note that these maps could be stored in a ClearCase attribute
     # on each project instead of hard-coded here in the trigger script
     # to give true per-project control.

     my %map_op_to_roles = (
        mkactivity => [ "projectmgr", "integrator", "developer" ],
        mkbl       => [ "projectmgr", "integrator" ],
        mkstream   => [ "projectmgr", "integrator", "developer" ],
     );

     # Which users belong to what roles?

     my %map_role_to_users = (
        projectmgr => [ "kate" ],
        integrator => [ "kate", "mike" ],
        developer  => [ "kate", "mike", "jones" ],
     );
```

```
    # Does user belong to any of the roles that can perform this operation?

    my ($role,$tmp_user);

    for $role (@{ $map_op_to_roles{$op} }) {
       for $tmp_user (@{ $map_role_to_users{$role} }) {
              if ($tmp_user eq $user) {
                     return 1;
              }
       }
    }

    return 0;
}

sub Main
{
    my $user = $ENV{CLEARCASE_USER};
    my $proj = $ENV{CLEARCASE_PROJECT};
    my $op   = $ENV{CLEARCASE_OP_KIND};
    my $pop  = $ENV{CLEARCASE_POP_KIND};

    my $perm = has_permission($user, $op, $proj);

    printf("$user %s permission to perform '$op' in project $proj\n",
       $perm ? "has" : "does NOT have");

    exit($perm ? 0 : 1);
}

Main();
```

2.7 Additional Uses for UCM Triggers

The examples shown in the previous sections represent just a few ways that you may use UCM triggers to enforce development policies. Other uses for UCM triggers include the following:

- Creating an integration between UCM and a change request management (CRM) system. Although we expect that most customers will use the out-of-the-box integration with ClearQuest, you may want to integrate with another CRM system. To accomplish this, you could do the following:

- Create a trigger type on **mkactivity** that creates a corresponding record in the CRM database when a developer makes a new activity.
- Create a trigger type on **setactivity** that transitions the record in the CRM database to a scheduled state when a developer starts working on an activity.
- Create a trigger type on **deliver** that transitions the record in the CRM database to a completed state when a developer finishes delivering the activity to the integration stream.

- Creating a trigger type on **rebase** that prevents developers from rebasing certain development streams. You may want to enforce this policy on a development stream that is being used to fix one particular bug.
- Creating a trigger type on **setactivity** that allows specific developers to work on specific activities.

3 Setting Up a ClearQuest User Database

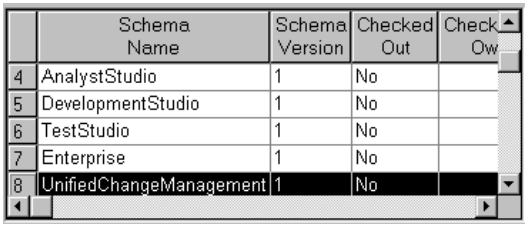
This chapter describes how to set up a ClearQuest user database so that you can use the UCM-ClearQuest integration for your project. The steps in this chapter are typically completed by the ClearQuest database administrator. ClearQuest includes predefined schemas that are ready for use with UCM. You can also enable a custom schema, or another predefined schema, to work with UCM. See *Managing Software Projects with ClearCase* for information on the decisions you need to make before setting up the integration.

3.1 Using the Predefined UCM-Enabled Schemas

The predefined UCM schemas, named **UnifiedChangeManagement** and **Enterprise**, include the record type, field, form, state, and other definitions necessary to work with a UCM project. To set up a ClearQuest user database to work with UCM:

1. Create a user database that is associated with one of the predefined UCM-enabled schemas. In the ClearQuest Designer, click **Database**>**New Database** to start the New Database Wizard.
2. Complete the steps in the wizard. Step 4 prompts you to select a schema to associate with the new database. Scroll the list of schema names and select the new schema, as shown in Figure 2.
3. Click **Finish**.

Figure 2 Associating a User Database with a UCM-Enabled Schema



	Schema Name	Schema Version	Checked Out	Check Ow
4	AnalystStudio	1	No	
5	DevelopmentStudio	1	No	
6	TestStudio	1	No	
7	Enterprise	1	No	
8	UnifiedChangeManagement	1	No	

3.2 Enabling a Schema to Work with UCM

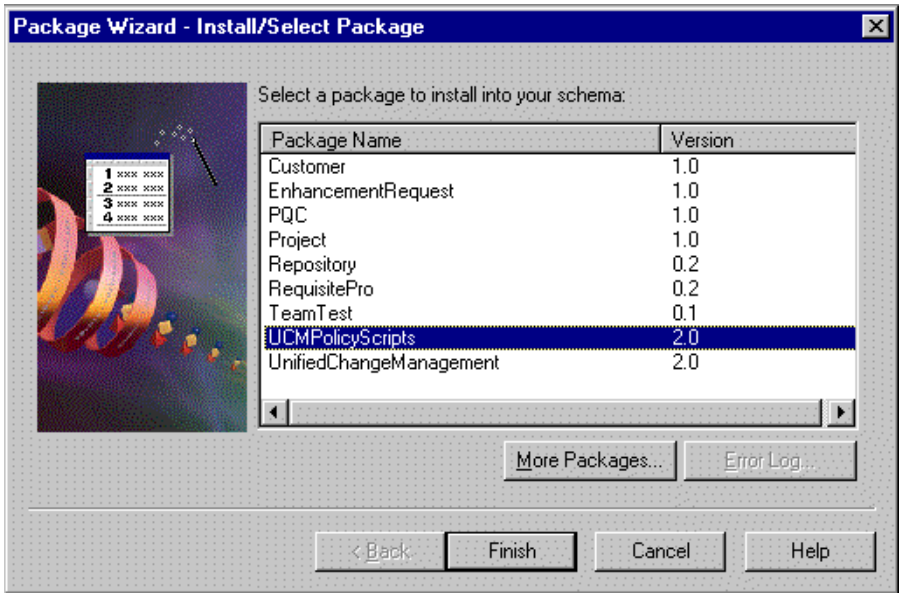
The predefined UCM schemas let you use the UCM-ClearQuest integration right away, but you may prefer to design a custom schema to track your project's activities and change requests, or you may prefer to use a different predefined schema. To enable a schema to work with UCM:

1. Ensure that the schema does not contain a record type named **UCM_Project**, which is a reserved name used by the UCM-ClearQuest integration.

2. In the ClearQuest Designer, click **Package**>**Package Wizard** to start the Package Wizard, as shown in Figure 3.

3. Add the **UCMPolicyScripts** package to your schema. If this package is not listed in the first page of the wizard, it has not been installed in your schema repository. To add the package to your schema repository, click **More Packages** to open the **Install Packages** dialog box; select the highest version of the package, and click **OK**. In the wizard, select the package, as shown in Figure 3. Click **Next**.

Figure 3 Adding the UCMPolicyScripts Package to a Schema



4. On the second page of the wizard, select your schema, and click **Finish**. To make the changes to the schema, ClearQuest checks out the schema for you. Check in the schema by clicking **File**>**Check In**. ClearQuest creates a new version of the schema.

5. Optionally, you can use the Package Wizard to apply the **BaseCMActivity** package to your schema. The **BaseCMActivity** package adds the BaseCMActivity record type to your schema. The BaseCMActivity record type is a lightweight activity record type. You may want to use the BaseCMActivity record type as a starting point and then modify it to include additional fields, states, and so on. If you want to rename the BaseCMActivity record type, be sure to do so before you create any records of that type.

6. Apply the **AMStateTypes** package to the schema. Start the Package Wizard. Select **AMStateTypes**, and click **Next**.

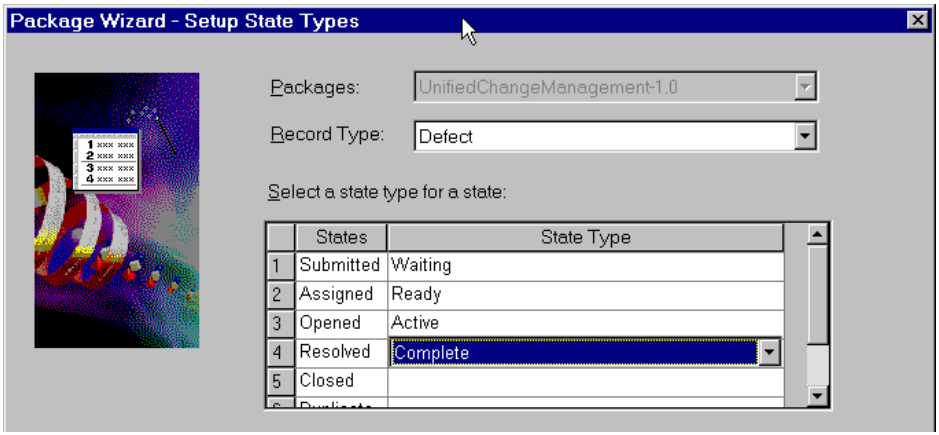
7. In the second page of the wizard, select your schema. Click **Next**.

8. The third page of the wizard prompts you to specify the schema's record types. Select the check boxes of the record types that you want to enable. Click **Next**. All selected record types must meet the requirements listed in *Requirements for Enabling Custom Record Types* on page 23.

9. In the fourth page of the wizard, you must assign state types to the states for each record type that you choose to enable. For each state, click in the adjacent state type cell to display the list of available state types, as shown in Figure 4, and select one. To enable another record type, click the arrow in the Record Type list to see the available record types. See *Setting State Types* on page 24 for a description of the four state types, and the rules for setting them.

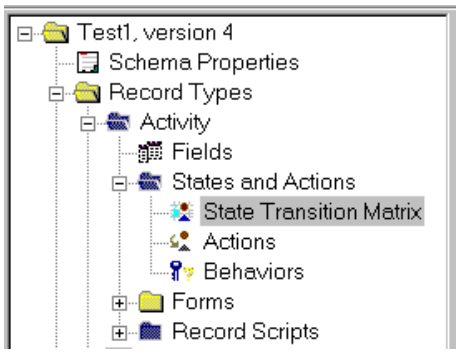
   When you are finished, click **Finish** to check out the schema.

Figure 4 Assigning State Types to a Record Type's States



10. Before you can check in your schema, you must set default actions for the states of each enabled record type. Default actions are state transition actions that ClearQuest takes when a developer begins to work on an activity or delivers an activity. In the ClearQuest Designer workspace, navigate to the record type's state transition matrix, as shown in Figure 5.

Figure 5 Navigating to Record Type's State Transition Matrix



Double-click **State Transition Matrix** to display the matrix. Right-click the state column heading, and select **Properties** from its shortcut menu. Click the **Default Action** tab. Select the default action. See *State Transition Default Action Requirements for Record Types* on page 24 for default action requirements. Before you can set default actions, you may need to add some actions to the record type. To do so, double-click **Actions** to display the Actions grid, and then click **Edit**>**Add Action**.

11. Validate the schema changes by clicking **File**>**Validate**. Fix any errors that ClearQuest displays, and then check in the schema by clicking **File**>**Check In**.

12. Apply the **UnifiedChangeManagement** package to the schema. Start the Package Wizard. Select **UnifiedChangeManagement**, and click **Next**.

13. In the second page of the wizard, select your schema. Click **Next**.

14. The third page of the wizard prompts you to specify the schema's record types. Select the check boxes of the same record types that you chose when you applied the **AMStateTypes** package. Click **Next**. All selected record types must meet the requirements listed in *Requirements for Enabling Custom Record Types* on page 23.

15. In the ClearQuest Designer workspace, navigate to the record type's **Behaviors**. Double-click **Behaviors** to display the Behaviors grid. Verify that the **Headline** field is set to **Mandatory** for all states. Verify that the **Owner** field is set to Mandatory for all Ready and Active state types.

16. Validate the schema changes by clicking **File**>**Validate**. Fix any errors that ClearQuest displays, and then check in the schema by clicking **File**>**Check In**.

17. Upgrade the user database so that it is associated with the UCM-enabled version of the schema by clicking **Database**>**Upgrade Database**. Alternatively, create a new user database that is based on the UCM-enabled version of the schema.

Requirements for Enabling Custom Record Types

Before you can apply the **UnifiedChangeManagement** package to a custom record type, the record type must meet the following requirements:

- It contains a field named **Headline** defined as a SHORT_STRING, and a field named **Owner** defined as a REFERENCE to the ClearQuest-supplied **users** record type. The **Headline** field must be at least 120 characters long.

- It does not contain fields with these names:

- **ucm_vob_object**
- **ucm_stream**
- **ucm_stream_object**
- **ucm_view**

- It contains an action named **Modify** of type Modify.

Setting State Types

The integration uses a state transition model to help you monitor the progress of activities. To implement this model, the integration adds state types to UCM-enabled schemas. Table 1 lists and describes the four state types. You must assign each state to a state type. You must have at least one state definition of state type Waiting, one of state type Ready, one of state type Active, and one of state type Complete.

Table 1 State Types in UCM-Enabled Schema

State Type	Description
Waiting	The activity is not ready to be worked on, either because it has not been assigned or it has not satisfied a dependency.
Ready	The activity is ready to be worked on. It has been assigned, and all dependencies have been satisfied.
Active	The developer has started work on the activity but has not completed it.
Complete	The developer has either worked on and completed the activity, or not worked on and abandoned the activity.

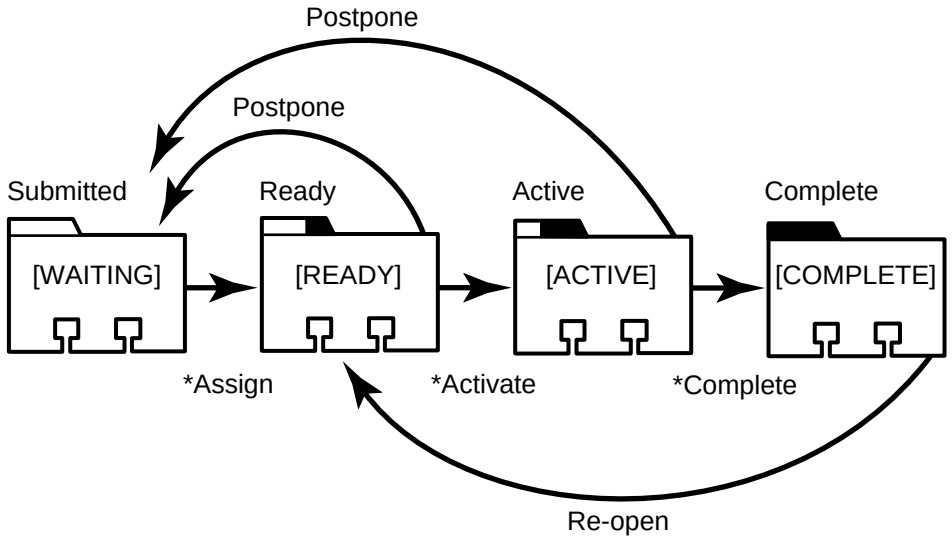
State Transition Default Action Requirements for Record Types

Record types can include numerous state definitions. However, UCM-enabled record types must have at least one path of transitions among state types as follows: Waiting to Ready to Active to Complete. The transition from one state to the next must be made by a default action.

For example, Figure 6 shows the actions and default actions between the states defined in the UCM-enabled BaseCMActivity record type included in the predefined UCM schema. The

default actions are identified with an asterisk (*). The state types are in uppercase letters enclosed in brackets. The states appear immediately above their state types.

Figure 6 State Transitions of UCM-enabled BaseCMActivity Record Type



In addition to this single path requirement, states must adhere to the following rules:

- All Waiting type states must have a default action that transitions to another Waiting type state or to either a Ready or Active type state.
- If a Ready type state has an action that transitions directly to a Waiting type state, that Waiting type state must have a default action that transitions directly to that Ready type state.
- All Ready type states must have a default action that transitions to another Ready type state or to an Active type state.
- All Ready type states must have at least one action that transitions directly to a Waiting type state.
- For the BaseCMActivity record type, its initial state must be a Waiting type.

3.3 Upgrading Your Schema to the Latest UCM Package

If you have a UCM-enabled ClearQuest schema from a previous release of ClearQuest, you may want to upgrade that schema with the latest version of the **UnifiedChangeManagement** package so that you can use new functionality. To upgrade the schema, perform the following steps:

1. In the ClearQuest Designer, click **Package**>**Package Wizard** to start the Package Wizard.
2. Select the **AMStateTypes** package, and click **Next**.
3. On the second page of the wizard, select your schema. Click **Next**.
4. The third page of the wizard prompts you to specify the schema's record types. Select the check boxes of the record types that you want to enable. Click **Next**.
5. On the fourth page of the wizard, you must assign state types to the states for each record type that you choose to enable. For each state, click in the adjacent state type cell to display the list of available state types (Figure 4) and select one. To enable another record type, click the arrow in the Record Type list to see the available record types. See *Setting State Types* on page 24 for a description of the four state types, and the rules for setting them.

   When you are finished, click **Finish** to check out the schema and exit the Package Wizard.
6. Validate the schema changes by clicking **File**>**Validate**. Fix any errors that ClearQuest displays, and then check in the schema by clicking **File**>**Check In**.
7. Start the Package Wizard again and select the new version of the **UnifiedChangeManagement** package.
8. On the second page of the wizard, select your schema. Click **Next**.
9. Click **Next** until you get to the last page of the wizard. Click **Finish**.
10. Validate the schema changes by clicking **File**>**Validate**. Fix any errors that ClearQuest displays, and then check in the schema by clicking **File**>**Check In**.
11. Upgrade the user database to associate it with the new version of the schema by clicking **Database**>**Upgrade Database**.

3.4 Customizing ClearQuest Project Policies

To implement the project policies, the integration adds the following pairs of scripts to a UCM-enabled schema:

- **UCM_ChkBeforeDeliver** and **UCM_ChkBeforeDeliver_Def**
- **UCM_ChkBeforeWorkOn** and **UCM_ChkBeforeWorkOn_Def**
- **UCM_CQActAfterDeliver** and **UCM_CQActAfterDeliver_Def**

Each policy has two scripts: a base script and a default script. The default scripts have **_Def** appended to their names and are installed by the **UnifiedChangeManagement** package. The integration invokes the base scripts, which are installed by the **UCMPolicyScripts** package. The base script calls the corresponding default script, which contains the logic for the default behavior. To modify the behavior of a policy, remove the call to the default script from the base script. Then add logic for the new behavior to the base script. Adhere to the rules stated in the base script.

Each script has a Visual Basic version and a Perl version. The Visual Basic scripts have a **UCM** prefix. The Perl scripts have a **UCU** prefix. For ClearQuest clients on Windows NT, the integration uses the Visual Basic scripts. For ClearQuest clients on UNIX, the integration uses the Perl scripts. If you modify a policy's behavior and your environment includes ClearQuest clients on both platforms, be sure to make the same changes in both the Visual Basic and Perl versions of the policy's script. Otherwise, the policy will behave differently for ClearQuest clients on UNIX and Windows NT.

For descriptions of these policies, see *Managing Software Projects with ClearCase*.

3.5 Associating Child Activity Records with a Parent Activity Record

As project manager, you may assign activities for large tasks to developers. When the developers research their activities, they may determine that they need to perform several separate activities to complete one large activity.

For example, an "Add customer verification functionality" activity may require significant work in the product's GUI, the command-line interface, and a library. To more accurately track the progress of the activity, you can decompose it into three separate activities.

By using the parent/child controls in ClearQuest, you can accomplish this decomposition and tie the child activities back to the parent activity.

Using Parent/Child Controls

In ClearQuest, you use controls to display fields in record forms. A parent/child control, when used with a reference or reference list field, lets you link related records. By adding a parent/child control to the record form of a UCM-enabled record type, you can provide the developers on your team with the ability to decompose a parent activity into several child activities.

To have ClearQuest change the state of the parent activity to Complete when all child activities have been completed, you need to write a hook. See *Administering Rational ClearQuest* for an example of such a hook.

3.6 Creating Users

Before you can assign activities to the developers on your project team, you must create user account profiles for each developer in ClearQuest. To do so:

1. In ClearQuest Designer, click **Tools**>**User Administration**.
2. Click **Add**.
3. Complete the **User Information** dialog box.

See *Administering Rational ClearQuest* and the ClearQuest Designer online help for details on creating user profiles.

3.7 Setting the Environment on UNIX

This section applies to UNIX only.

Before you can enable a UCM project to work with a ClearQuest user database, you must define two environment variables as shown in Table 2. Developers who want to use the integration must also define these variables on their machines.

The ClearQuest installation directory includes a C shell script, **cq_setup.csh**, which you can execute to set the environment variables for you. For example:

% **source** *ClearQuest-install-directory***/cq_setup.csh**

Table 2 Environment Variables Required for Integration

Variable	Setting
$CQ_HOME	*ClearQuest-install-directory*/releases/ClearquestClient
$LD_LIBRARY_PATH (**$SHLIB_PATH** on HP-UX)	Must include: *ClearCase-install-directory*/shlib and *ClearQuest-install-directory*/releases/ClearquestClient/*architecture*/shlib

In addition, if you have multiple ClearQuest schema repositories, you must set the **$SQUID_DBSET** environment variable to the name of the schema repository you want to use.

3.8 How MultiSite Affects the UCM-ClearQuest Integration

If you use MultiSite to replicate the PVOB or ClearQuest user database involved in the UCM-ClearQuest integration, you need to be aware of several requirements. This section describes those requirements.

Replica and Naming Requirements

When you set up the UCM-ClearQuest integration, you establish a link between a PVOB and a ClearQuest user database. If you use MultiSite, the following requirements apply:

- Each site that contains a linked PVOB replica must contain a replica of the ClearQuest user database to which the PVOB is linked. Similarly, each site that contains a linked ClearQuest user database replica must contain a replica of the PVOB to which the user database is linked.

- The name of the linked ClearQuest user database replica must match the name of the linked PVOB replica at the same site.

Enabling a Project to Use the UCM-ClearQuest Integration

This section describes the additional steps to set up the UCM-ClearQuest integration when you use MultiSite. For the full set of steps required to enable a project to work with ClearQuest, see the *Setting Up the Project* chapter in *Managing Software Projects with ClearCase*.

Transferring Mastership of the PVOB's Root Folder

The first time you enable a project within a PVOB to work with ClearQuest, your current PVOB replica must master the PVOB's root folder. If your current replica does not have mastership, transfer mastership of the root folder by using the **multitool chmaster** command at the replica that masters the root folder. The following example transfers mastership of the root folder from the current replica to the **lowell** replica.

multitool chmaster lowell folder:RootFolder

See *ClearCase MultiSite Manual* for details on transferring mastership.

Transferring Mastership of the Project

Before you enable a project to work with ClearQuest, your current PVOB replica must master the project. If your replica does not master the project, transfer mastership of the project by using the **multitool chmaster** command at the replica that masters the project.

When you enable the project to work with ClearQuest, the integration creates a corresponding project record in the ClearQuest user database and assigns mastership of that record to the current replica of the ClearQuest user database. If a project record with the same name as the project exists in the ClearQuest user database when you enable the project, and that project record is not mastered by your current replica, you must transfer mastership of the project record to your current replica.

Linking Activities to ClearQuest Records

If a project contains activities, when you enable that project to work with ClearQuest, the integration creates corresponding ClearQuest records for the activities and links the records to the activities. The integration cannot link activities that are mastered by remote replicas. To link activities that are mastered by a remote replica:

1. At the remote site, start ClearCase Project Explorer. On UNIX, enter **clearprojexp**. On Windows, in the left pane of ClearCase Explorer, click **UCM**, and then click **Project Explorer**.
2. In the Project Explorer, display the project's property sheet, and click the **ClearQuest** tab.
3. Click **Ensure all Activities are Linked**. The integration checks all the project's activities. If the project is enabled, the integration links any unlinked activities. The integration then displays the following summary information:
   - Number of activities that had to be linked.
   - Number of activities that were previously linked.
   - Number of activities that could not be linked because they are not mastered by the current PVOB replica. In this case, the integration also displays a list of replicas on which you must run the **Ensure all Activities are Linked** operation again to correct the problem.
4. At each replica on the list described in Step #3, repeat Step #1 through Step #3.

Managing the Project

This section describes how MultiSite affects how you maintain the project after enabling it to work with ClearQuest.

Changing Project Policy Settings

Before you can change a project's policy settings from within ClearQuest, the ClearQuest project record must be mastered. Similarly, before you can change a project's policy settings from within ClearCase, the project object must be mastered. After you change a project's policy settings in the current replica, the new settings do not take effect in streams in sibling replicas until you synchronize the current replica with those replicas. See *ClearCase MultiSite Manual* for details on synchronizing replicas.

Controlling Deliver Operations

The Do ClearQuest Action After Delivery project policy transitions activities to a Complete type state when a deliver operation completes successfully. For this policy to work correctly in a MultiSite environment, the activities being delivered must be mastered by the same replica that masters the target integration stream. To ensure that this is the case, you can set the Check Mastership Before Delivery policy.

The behavior of the Check Mastership Before Delivery policy depends on whether the deliver operation is local or remote. If the deliver operation is local, meaning that the target integration stream is mastered by the local PVOB replica, this policy causes the deliver operation to fail unless all activities being delivered are mastered locally.

A *remote deliver* operation is one for which the target integration stream is mastered by a remote PVOB replica. The developer starts the deliver operation but ClearCase leaves the operation in a *posted* state. The project manager at the remote site completes the deliver operation.

For a remote deliver operation, the Check Mastership Before Delivery policy causes the following behavior:

- If all activities in the deliver operation are mastered by the remote replica, ClearCase allows the deliver operation to proceed.
- If the deliver operation contains activities that are mastered by the local replica, MultiSite transfers mastership of those activities to the remote replica. After the project manager at the remote site performs any required merges and completes the deliver operation, MultiSite transfers mastership of the activities back to the local replica.
- If the deliver operation contains activities that are mastered by a third replica, the deliver operation fails.

Changing the Project Name

The integration links a project name to the title field in the corresponding ClearQuest project record. If you change the project name in ClearCase, the integration makes the same change to the title field in the corresponding ClearQuest project record. Similarly, if you change the title in ClearQuest, the integration makes the same change to the project name in ClearCase. Before you can change the project name and title in a MultiSite environment, the project record and the project object must both be mastered.

Working on Activities

Before you can work on, set, or change an activity, the activity object and its ClearQuest record must be mastered locally.

4 Cross-Platform File Access

In mixed networks of Windows and UNIX computers, certain ClearCase operations may require a Windows or UNIX computer to access the file system of a different type of computer. ClearCase supports several protocols that allow a Windows computer to access the file system of a UNIX computer and also provides for more limited access by UNIX computers to ClearCase data in Windows file systems. The following protocols—some of which can only be enabled using a software from a third party (neither Rational nor the computer vendor)—may be used.

- **ClearCase File Server (CCFS)** — The ClearCase File Server is a TCP/IP-based file transfer mechanism included with ClearCase. It provides snapshot views with access to VOB data. It does not support dynamic views. For information about CCFS, see *ClearCase File Server* on page 35.

- **NFS client products** — NFS client products for Windows NT are available from several vendors. These products allow Windows NT computers to access UNIX file systems using the NFS protocol, which all UNIX computers support. You install an NFS client product on each Windows NT computer from which you want to access UNIX VOBs and views. For more information about which NFS client products ClearCase supports and how to configure them, see *NFS Client Products* on page 36.

- **SMB server products** — The SMB (Server Message Block) protocol is the native protocol that Windows computers use for network file-system access. SMB servers that run on UNIX computers allow Windows computers to access UNIX VOBs and views using native Windows protocols. You install an SMB server product on each UNIX VOB or view server you will access from a Windows NT client. For more information about which SMB server products ClearCase supports and how to configure them, see *SMB Server Products* on page 43

Table 3 lists the protocols that ClearCase clients can use to access VOB data. Table 4 lists the protocols that ClearCase clients use to access **view_server** storage. In these tables, protocols

native to their respective computer platforms are labeled “Native.”CCFS is included in ClearCase. All other protocols require third-party software support.

Table 3 Protocols for ClearCase Client Access to VOB Data

Client Platform	Access to VOB data on UNIX	Access to VOB data on Windows NT
Windows 98, Windows Me	CCFS	Native SMB
Windows NT (dynamic views)	Third-party NFS or SMB	Native SMB
Windows NT (snapshot views)	CCFS, third-party NFS or SMB	Native SMB
UNIX (dynamic views)	Native NFS	Unsupported
UNIX (snapshot views)	Native NFS	CCFS

Table 4 Protocols for ClearCase Client Access to View Data

Client Platform	Access to view data on UNIX	Access to view data on Windows NT
Windows 98, Windows Me	See note.	Native SMB
Windows NT (dynamic views)	Third-party NFS or SMB	Native SMB
Windows NT (snapshot views)	Third-party NFS or SMB	Native SMB
UNIX (dynamic views)	Native NFS	Unsupported
UNIX (snapshot views)	Native NFS	See note.

NOTE: When a view has been created with the **-ngpath** option and both the client and server platforms are running ClearCase 4.1 or later, Windows 98 and Windows Me platforms can access view data on UNIX, and UNIX snapshot views can access view data on Windows NT using the native ClearCase RPC mechanism. This configuration is found most often in ClearCase LT communities.

Table 5 lists the protocols used by a view_server process to access VOB data.

Table 5 Protocols for view_server Access to VOB Data

view_server platform	Access to VOB data on UNIX	Access to VOB data on Windows NT
Windows NT	CCFS, third-party NFS, or SMB	Native SMB
UNIX	Native NFS	CCFS

4.1 ClearCase File Server

The ClearCase File Server is a TCP/IP-based mechanism that enables cross-platform file transfers between VOB servers and snapshot views. It supports access by snapshot views on Windows computers to VOB data on UNIX computers and access by snapshot views on UNIX computers to VOB data on Windows NT.

If a ClearCase client uses only snapshot views, no third-party NFS client-based or SMB server-based product is necessary to access VOB data on any platform. Dynamic views support access by snapshot views on Windows computers to VOB data on UNIX computers. Dynamic views on Windows NT computers still require a supported NFS client or SMB server product to access UNIX VOBs.

ClearCase clients running Windows Me or Windows 98 use CCFS as their sole transfer mechanism when accessing UNIX VOBs. Because Windows Me and Windows 98 computers cannot use dynamic views or run view servers, you must have at least one Windows NT computer that will run the view server process and contain the view storage directory for snapshot views created on Windows Me and Windows 98 computers.

When CCFS is enabled, file transfers between snapshot view clients and VOB servers (for example, operations such as checking out, checking in, and creating and updating snapshot views) take place over a standard TCP/IP connection. File transfers between a VOB server and a snapshot view's view server also use this TCP/IP connection.

CCFS is always enabled on a UNIX computer running ClearCase. It may be enabled or disabled on Windows NT using the **ClearCase** program in Control Panel

Enabling and Disabling CCFS on Windows NT

To enable and disable CCFS on a Windows NT computer:

1. Click **Start**>**Settings**>**Control Panel**. Open the **ClearCase** program.
2. On the **Options** tab, select the **Use CCFS to access UNIX VOBs** check box to enable use of CCFS. Clear this check box to disable use of CCFS.

   Click **OK**.
3. Shut down and restart the computer to ensure that the change takes effect for all processes.

NOTE: When CCFS is disabled (which is the default setting on Windows NT), you must have a supported NFS client or SMB server product to access UNIX VOBs.

4.2 NFS Client Products

ClearCase supports these NFS client products on Windows NT computers:

- Microsoft Windows NT Services for UNIX Client for NFS Products (SFU 1.0)
- Intergraph DiskAccess
- Hummingbird NFS Maestro

If you are using an NFS client product, you must install it on each Windows NT client that will access VOBs or views located on UNIX servers. You must install the product correctly and completely; in particular, you must assign and configure the NFS daemon and authentication process.

NOTE: The *READ ME FIRST* chapter in *ClearCase and MultiSite Release Notes* (Windows edition) contains last-minute information about NFS client products, including which versions of those products ClearCase supports.For more information about configuring NFS client products, read the remainder of that manual.

The rest of this section describes configuration procedures specific to using an NFS client product with ClearCase. *Read and perform all procedures recommended for your product.*

Disabling Automatic Case Conversion

Some NFS client products change the case of file names by default, typically by converting to lowercase. Because ClearCase is case-sensitive, you need to disable case conversion, as described later in this section.

NOTE: Typically, you can use a command-line option to disable case conversion for a particular NFS mount. However, ClearCase can *automount* remote storage directories. See *Automounting and NFS Client Software* on page 39. For correct behavior on these mounts, configure NFS mount drive options to disable case conversion.

Microsoft SFU and Intergraph DiskAccess

To disable automatic case conversion:

1. Start the Client for NFS (for SFU) or DiskAccess (for DiskAccess) Control Panel program.
2. On the **Filenames** tab, click **Preserve Case (no conversion)**.

Hummingbird NFS Maestro

To disable automatic case conversion:

1. Click **Start**>**Settings**>**Control Panel**. Start the Network program.
2. On the **Services** tab, select **NFS Maestro for Windows NT Client**.
3. Click **Properties** to open the client configuration dialog box.
4. Under **Filename Capitalization**, click **Preserve Case**.

Setting an NFS Client's Default Protection

If you plan to work in a shared UNIX view, configure your NFS client with a default protection that grants group write access. Without this permission, other developers cannot modify view-private files that you have created.

Microsoft SFU or Intergraph DiskAccess

To set the default protection:

1. Start the Client for NFS (for SFU) or DiskAccess (for DiskAccess) Control Panel program.
2. On the **File Access** tab, ensure **User** is `RWX`, **Group** is `RWX`, and **Other** is `RX`.

Hummingbird NFS Maestro

To set the default protection:

1. Click **Start**>**Settings**>**Control Panel**. Start the Network program.
2. On the **Services** tab, select **NFS Maestro for Windows NT –Client**.
3. Click **Properties** to open the client configuration dialog box.
4. Under **Default Protection**, specify the protections as follows:

```
User Group Other
RWX  RWX   RWX
xxx  xxx   x x
```

Setting the Correct Logon Name

To avoid VOB and view access permission problems, do not log on to an NFS server as user **nobody** or with any user or group ID that does not match your Windows NT user and primary group IDs.

To verify that your Windows NT user name/UID and group name/GID match their UNIX counterparts, pass the name of a UNIX NFS server to *ccase-home-dir***etc\utils\credmap**. For example:

*ccase-home-dir***etc\utils\credmap saturn**

After you confirm your user name/UID and group name/GID, supply the user name as an NFS logon parameter.

Microsoft SFU or Intergraph DiskAccess

To set your logon user name:

1. Start the Client for NFS (for SFU) or DiskAccess (for DiskAccess) Control Panel program.
2. On the **Authentication** tab, type the correct **User Name**, **Password**, and **PCNFSD Server**.

3. Click **OK**; your logon session is validated.

Hummingbird NFS Maestro

To set your logon user name; at the command prompt, run the **nfs register** command:

nfs register *username*

This command prompts for a password.

Hummingbird NFS Maestro: Disabling DOS Sharing

The Maestro **DOS Sharing** option is incompatible with ClearCase use. When using Hummingbird NFS Maestro with ClearCase, you must disable this mode. If you do not, you may encounter MVFS log errors when attempting to open MVFS files. For example:

```
ZwOpenFile returned status 0xc0000043
```

This error indicates a sharing violation.

To disable DOS Sharing:

1. Start the Network program in Control Panel.
2. On the **Services** tab, select **NFS Maestro for Windows NT –Client**.
3. Click **Properties** to open the client configuration dialog box.
4. Under **Default Links**, clear the **DOS-Style Sharing** check box.

Automounting and NFS Client Software

When you mount a UNIX VOB or start a UNIX dynamic view, ClearCase needs to access the *VOB storage directory* or *view storage directory* on the UNIX file-system partition. In the ClearCase program in Control Panel, the setting of the **Enable automatic mounting of NFS storage directories** check box determines how ClearCase accesses those directories when you use NFS client products.

All supported NFS client products can process UNC names. If you are using one of these products, clear the **Enable automatic mounting of NFS storage directories** check box. ClearCase then uses UNC names to access UNIX VOB and view storage directories. We recommend that you configure ClearCase hosts in this way.

If you have been using ClearCase with the **Enable automatic mounting of NFS storage directories** check box selected, you can continue to do so. ClearCase then maps Windows drive letters to UNIX VOB and view storage directories and accesses the directories through those drive letters.

NOTE: These drive letters are for internal ClearCase use. They are different from the drive letters you can assign and use for your own work within views. In particular, do not confuse them with the drive letters you can assign to dynamic views when you start those views.

We recommend that you disable automounting when using any supported NFS client product. If you install Microsoft SFU or Intergraph DiskAccess before you install ClearCase, the ClearCase installation procedure disables automounting for you.

If you are using Hummingbird NFS Maestro or if you install a supported NFS product after you have installed ClearCase, you can disable automounting using the ClearCase program in Control Panel.

1. Click **Start**>**Settings**>**Control Panel.** Open the ClearCase program.
2. On the **Options** tab, clear the **Enable automatic mounting of NFS storage directories** check box.
3. Click **OK**.

Microsoft SFU or Intergraph DiskAccess: Setting Up the ClearCase Server Process User and ClearCase Group

SFU and DiskAccess use a scheme to map Windows NT user credentials to NFS user and group ID. This scheme requires that you perform an additional setup procedure for the special ClearCase server process user account, as described in *Administering ClearCase*.

The setup procedure enables ClearCase services to access remote NFS view and VOB storage directories with the proper privileges, which enables operations such as cleartext construction.

Cleartext construction is the process by which data is extracted from the VOB storage area source container by a ClearCase type manager. This type manager then constructs or creates a cleartext

container the first time a version is accessed. For faster access, subsequent reads of that version access the cleartext file directly.

Setting Up the UNIX Account

On UNIX, the group ID (GID) is used for the permission to construct cleartext. The UNIX primary group name must match the user's Windows NT primary group name.

NOTE: By convention, the name of the ClearCase group is **clearcase**. A community can choose a different name as long as all ClearCase hosts in the community define the ClearCase group as that group name. If multiple communities share a single Windows NT domain, each community must have a unique name for its ClearCase group.

As part of setting up SFU or DiskAccess, the administrator needs to supply the ClearCase server process user with a valid UNIX name and password for a user account that belongs to the correct primary group. The administrator can identify the UNIX account by performing one of the following tasks:

- Add a UNIX account that matches the Windows NT name of the ClearCase server process user account.
- Add a new UNIX account that does not match the Windows NT name.
- Use an existing UNIX account.

Preparing the Windows NT Client

Set up the ClearCase server process user account (**clearcase_albd**) on every ClearCase Windows NT client that will access remote NFS views or VOBS. For the first client, follow the steps below. For subsequent clients, either follow the steps here or perform the steps in *Alternative Setup: Administrative Option*.

1. Install SFU or DiskAccess, restart, and set up your regular user account, from which you will log on to SFU or DiskAccess.
2. Log off from your current user session.
3. Log on as the ClearCase server process user on your system.
4. Start the Client for NFS (for SFU) or DiskAccess (for DiskAccess) Control Panel program.
5. On the **Authentication** tab, specify the UNIX name and password that you established in *Setting Up the UNIX Account*.

6. Click **OK**.

   When you exit the Client for NFS (for SFU) or DiskAccess (for DiskAccess) program, read the confirmation dialog box. Verify that you are currently logged on with the desired UID and GID.

   If there is an error, or if you are logged on with UID and GID of -1 and -2, repeat Step #3 through Step #6 in this section.

7. Log off.

Alternative Setup: Administrative Option

Instead of performing the steps in *Preparing the Windows NT Client*, you can instead perform the steps in this section. For example, it may be inconvenient to log on as the ClearCase server process user on every client system. As an alternative, perform the following steps:

1. Log on as the ClearCase server process user (**clearcase_albd**) on one system.

2. Run the Windows Registry Editor (type **regedit** in the **Run** dialog box or from a command prompt).

3. Carefully save the *security-id* key here:

   HKEY_LOCAL_MACHINE\SOFTWARE\Intergraph\DiskAccess\CurrentVersion\Users

   Select the *security-id* subkey whose **Name** key value is your ClearCase server process user account name. To find the correct subkey, type the following command, and look on the line in the output that starts with `SID`:

   *ccase-home-dir***etc\utils\creds clearcase_albd**

   NOTE: If your community uses a ClearCase server process user account name other than **clearcase_albd**, use that name instead.

4. Save the key to a registry file by clicking **Registry**>**Export Registry File** in the Registry Editor.

   To share the registry file, mail it to users or place it on a shared directory. When users want to load the file into the registry, they double-click the file name in a file browser such as Windows Explorer. The system displays a confirmation message after it loads the file.

   All users who use the same ClearCase server process user account and the same primary Windows NT group that maps to the UNIX GID to which it was logged on can share this file.

Microsoft SFU: Configuring the Default LAN

Browsing of UNIX resources from Windows NT is not enabled by default on SFU. You must enable this feature manually for it to work. Otherwise, users cannot browse for UNIX resources using either **net view** at the command prompt or **Network Neighborhood** on the desktop. After configuring the client for NFS, take the following steps:

1. Start the **Client for NFS** Control Panel program.
2. On the **Configured NFS LANs** tab, click **Add**, enter a name for the LAN (for example, **Default_LAN**), click **Specify LAN to browse**, and then click **OK**.
3. On the **Configured NFS LANs** tab, click **Edit**, and in the **Broadcast Address** box type **255.255.255.255**. You can either accept the defaults or customize the rest of the broadcast parameters.
4. Click **OK** in the **Broadcasting** dialog box; then click **OK** in the **Client for NFS** dialog box.

NOTE: Configuring SFU for browsing as described here is not required by ClearCase and increases the UDP broadcast traffic on your network.

4.3 SMB Server Products

Rational supports two SMB products— Samba version 2.2 (available from **www.samba.org**) and Syntax TotalNET Advanced Server (TAS)—to enable access to UNIX file systems from Windows NT computers. This section describes how to install and configure Samba and TAS.

The *READ ME FIRST* chapter in *ClearCase and MultiSite Release Notes* (Windows edition) contains last-minute information about SMB server products, including which versions of those products that ClearCase supports. For more information about configuring SMB server products, read the remainder of that manual. For information on current UNIX platform support for ClearCase, Samba, and TAS, see the Rational ClearCase Web page.

Installing and Configuring Samba 2.2

ClearCase supports use of Samba 2.2 to provide Windows NT computers that use dynamic views with access to VOBs and views on Solaris 8 or later.

Samba 2.2 can be downloaded from **www.samba.org**. Download it and follow the installation instructions for the operating system on which you are installing it. Samba must be installed and configured on each UNIX VOB and view server that you want to access from Windows NT.

To configure Samba for use by ClearCase, you must do the following:

1. Create a Samba username map for the **clearcase_albd** user
2. Configure Samba globals
3. Create shares for VOB and view storage
4. Start Samba services

Creating a Samba Username Map for clearcase_albd

NOTE: In this section, we assume that the user account for the ClearCase server process on Windows NT is named **clearcase_albd**. If your user account for this server process is configured to use a different name, use that name instead.

Samba requires a username map that associates the user account for ClearCase server process Windows NT with a UNIX user account. For more information on administering Windows NT domains, see the release 4.1 edition of *Administering ClearCase*

To create the Samba username map, use any text editor to create a file named **username.map** on the host where Samba is installed. We recommend that you create the file in the same directory where you have installed other Samba configuration files (such as **smb.conf**).

The file must contain a line of the form

```
account = clearcase_albd
```

where `account` is the name of an existing UNIX user account. We strongly recommend that this user's primary group (the group listed in the user's entry in the passwd database) be one to which all ClearCase users accessing VOBs and views on this server belong. For details about group- and user-level access to ClearCase data, see the chapter on access controls in the release 4.1 edition of *Administering ClearCase*

For more information about the **username.map** file, see the Samba documentation.

Using the Samba Web Administration Tool (SWAT)

Samba can be configured using various methods that range from a simple text editor to graphical tools. The examples in this document describe the configuration of Samba through the use of the Samba Web Administration Tool (SWAT), which is included in the Samba download. Instructions included with the Samba download explain how to enable this tool.

To access the SWAT interface:

1. Type a URL of this format in a Web browser:

   http://*computer:port#*

   where *computer* is the host name of a UNIX VOB server or view server host on which you have installed Samba and *port#* represents the SWAT port number. (The default value is 901.)

2. Log on as **root.** The SWAT interface now appears in your browser.

Configuring Samba Globals for ClearCase

Click the GLOBALS icon at the top of the SWAT interface's home page. Then click **Advanced View**. Set the global options as described in Table 6.

Table 6 Samba Global Settings for ClearCase (Part 1 of 2)

Base options	
workgroup	Set to the name of the Windows NT domain to which ClearCase hosts accessing this server belong
netbios name	Set to the host name of this computer
Security options	
security	DOMAIN (recommended) or USER (see note)
encrypt passwords	Yes
create mask	0775
directory mask	0775
username map	Set to the local pathname of the **username.map** file

Table 6 Samba Global Settings for ClearCase (Part 2 of 2)

Locking options	
oplocks	No
kernel oplocks	No
File-name handling options	
case sensitive	No
preserve case	Yes

NOTE: If you select USER security, you must enter every user that will access Samba file services in a local password encryption database on the server that supports those file services. Click the PASSWORD icon on the SWAT home page. In the Server Password Management section, enter the name and password of each user.

ClearCase has no special requirements for other Samba globals, so you may configure them in any way that's appropriate for your site.

Creating Shares for VOB and View Storage

You must create one or more Samba shares to hold server storage locations or individual VOB or view storage directories. To create a Samba share:

1. Click the SHARES icon at the top of the SWAT interface's home page.
2. Enter a name for the share in the text field to the right of the **Create Share** button. To simplify administration, we recommend that the share name be similar or identical to that of the UNIX directory whose name you will enter in Step #4.
3. Click **Create Share**.
4. Edit the path option under **Base Options**. Set its value to be a directory under which the VOB or view storage areas reside. The VOB or view storage areas do not need to be in the directory specified, but they must be somewhere below the specified directory.
5. Click **Commit Changes**.

Starting Samba Services

The Samba **smbd** and **nmbd** services must be running before Windows computers can access files using Samba. We recommend that you configure your UNIX host to start the **smbd** and **nmbd** services at boot time. Platform-specific instructions for configuring automatic service startup are included in the Samba documentation.

Samba services can also be started manually from the SWAT interface using the following procedure:

1. Click the STATUS icon at the top of the SWAT interface's home page.
2. Click **Start smbd**.The page refreshes and should display the **smbd** status as `running`.
3. Click **Start nmbd**.The page refreshes and should display the **nmbd** status as `running`.

Configuring ClearCase to Support Samba

For all ClearCase clients on Windows NT that have the MVFS installed and that will access Samba shares, change the **MVFS Performance** settings in the ClearCase program in Control Panel as follows:

1. Click **Start**>**Settings**>**Control Panel**. Start ClearCase.
2. On the **MVFS Performance** tab:
   - Select **Override** for both **Maximum number of mnodes to keep on the free list** and **Maximum number of mnodes to keep for cleartext free list**.
   - Set the value for both to 800.
3. Click **OK** to apply the changes and close the dialog box.
4. Restart Windows NT.

Testing the Samba Configuration on Non-ClearCase Files

We recommend that you test the Samba installation and configuration using non-ClearCase files and directories before attempting to use Samba to provide file access to VOBs and views, as follows:

1. Create a directory on your Samba server (for example, **/testshare/testdir**) and a test file in that directory (for example, **/testshare/testdir/testfile**).

2. Create a Samba share using **testshare** as the share name and **/testshare** as the path name for the share.

3. From a Windows NT client, create a file in the Samba share. Then verify that the UNIX user and group settings for that file are correct.

4. Verify that all Windows NT clients can access the Samba share, including testing permission and access restrictions, until you are confident that Samba is working properly.

Testing the Samba Configuration with ClearCase

To verify that ClearCase and Samba are working together properly:

1. On a UNIX VOB or view server, install and configure Samba as described in this chapter, creating shares for VOB and/or view storage.

2. Verify that your ClearCase user and group assignments are appropriate. To do so, refer to the chapter on this subject in the release 4.1 edition of *Administering ClearCase*.

3. Verify that you can access VOBs and views on the server from a UNIX client.

4. Log on to a ClearCase client on Windows NT. Use the Region Synchronizer to import VOB and view tags for VOBs and views hosted on the UNIX server into the Windows region.

5. Ensure that you can use these views and VOBs by performing some basic ClearCase operations (for example, **mkelem**, **checkin**, and **checkout**) in them.

Syntax TotalNET Advanced Server

ClearCase supports the Syntax TotalNET Advanced Server (TAS) SMB server product to provide Windows NT computers using dynamic views with access to VOBs and views on any of the following UNIX Platforms:

- Solaris 2.5.1 or later
- HP-UX 11.0 or later
- AIX 4.3 or later

Installing TAS 6.0

This section describes how to install TAS 6.0, including how to configure TAS and ClearCase to support mixed-environment file access. If you are using Syntax TotalNET Advanced Server, you must install and configure it on each UNIX VOB and view server that you want to access from a Windows NT client.

Follow the instructions in the appropriate platform-specific installation section of *TotalNET Advanced Server Release Notes* to install TAS on each VOB and view server requiring access from Windows NT.

Enabling the Multiuser Kernel Driver on AIX

If you are installing Syntax TotalNET Advanced Server on an AIX platform, you must enable the multiuser kernel driver after installing TAS. This step provides support for the TAS SMB multiplexor, which is required when using ClearCase with TAS on AIX.

To enable the multiuser kernel driver, use the TAS **smbmxenable** command. This command does not take any command-line options or arguments.

cd /var/totalnet/usr/sbin
./smbmxenable

To disable the multi-user kernel driver, use the TAS **smbmxdisable** command. This command does not take any command-line options or arguments.

cd /var/totalnet/usr/sbin
./smbmxdisable

NOTE: You cannot enable or disable the multiuser support from the Framework interface. You for details about multiuser support on AIX platforms, see Appendix E, *TAS Multiplexing*, in the *TotalNET Advanced Server Administration Manual.*

Accessing the Syntax Administration Framework

You can configure and administer TAS using the Syntax Administration Framework (formerly known as the TotalNET Administration Suite, or TNAS) Web interface. For details, see the chapter on syntax administration framework in *TotalNET Advanced Server Administration Manual*.

To access the Syntax Administration Framework Web interface:

1. Type a URL of this format in a Web browser:

http://*computer***:***port#*

where

- *computer* is the host name of a UNIX VOB- or view-server host on which you have installed TAS
- *port#* represents the Framework port number (the default is 7777)

The Syntax Enterprise Services page appears.

2. Click **Syntax Administration Framework**; a Framework logon program appears.
3. Log on as **root**, using the **root** password for the TAS server. The Framework interface now appears in your browser.
4. Click **TAS Configuration and Administration** in the *sphere frame* (that is, the frame at the upper right of the interface).

   The TAS configuration and administration menu now appears in the *menu frame* (that is, the frame at the lower left of the interface).

Performing Initial Setup of TAS

NOTE: If you are upgrading an existing installation of TAS, the upgrade procedures preserve the previous configuration, including existing TAS volumes and file services supporting ClearCase, so you can skip the remaining sections of this chapter. After you have upgraded, ensure that opportunistic locks are disabled for each TAS volume that contains ClearCase storage. (The **Support opportunistic locks** check box in the volume definition should be cleared.) For details, see *Administering Volume Attributes* in the chapter on volume administration in *TotalNET Advanced Server Administration Manual*.

After you have installed TAS on a server, you must perform an initial setup on that TAS installation. For details, see the chapter on initial setup in *TotalNET Advanced Server Administration Manual*.

Click **Initial Setup** in the menu frame of the Framework Web interface, and follow the instructions in the Syntax documentation, subject to the changes noted in these sections that are specific to use of TAS with ClearCase.

General TAS Settings

Accept the defaults for **Admin user**, **Admin group**, and so on in the **General TAS Settings** pane.

Enabling and Configuring the CIFS Realm

In the **Select Realms to Configure** pane, enable the CIFS realm, and click **Next**; the **CIFS Realm Configuration** pane appears.

NOTE: ClearCase does not require that the NetWare and AppleTalk realms be enabled.

Configure the CIFS realm as follows:

- **Server name** — Type the name of the VOB or view server, if it is not already the default.
- **Workgroup** — Type the name of the Windows NT domain to which your ClearCase clients belong.
- **Transports** — Select the protocols appropriate for your site.
- **Device for NetBEUI** — Accept the default.
- **WINS Server(s)** — If you are using proxy server authentication mode for CIFS file services (see *Configuring the File Service* on page 53), you may have to specify the IP addresses of the WINS servers for the network on which the authentication proxy server resides.

For details about configuring the CIFS realm, see the section *Updating CIFS Realm Configuration* in the chapter *Updating Realm Configuration* in the *System/Realm Administration* of the *TotalNET Advanced Server Administration Manual*.

Configuring TAS to Support ClearCase

After initial setup, configure the TAS server to support ClearCase, using the Framework Web interface.

Creating a TAS Username Map for clearcase_albd

Create a TAS username map from the user account for the ClearCase server process on Windows NT to a UNIX user account whose primary group ID (GID) can access all VOBs and views that will be accessed by TAS file services. In this section, we assume that this user account is named **clearcase_albd**. If the user account for your server process is configured to use a different name, use that name instead.

To create the TAS username map:

1. Click **TAS System** in the menu frame; the **TAS System Configuration and Administration** pane appears.

2. Click **Username Maps**; the **Username Maps** pane appears. Make these changes to support ClearCase:

   - In the text box, type the name of an existing UNIX user account and click **Create**. We strongly recommend that this user's primary group (the group listed in the user's entry in the *passwd* database) be one to which all ClearCase users who access VOBs and views on this server belong.

     For details about group- and user-level access to ClearCase data, see *Administering ClearCase*

   - In **List of client accounts**, type **clearcase_albd**.

   Click **Submit** at the bottom of the form; then click **OK** in the confirmation message.

For details about the ClearCase server process user, see the chapter on administering WIndows NT domains in *Administering ClearCase*. For details about creating user name mappings in TAS, see *Username Maps* in the *Sharing Volumes and Printers* chapter of *TotalNET Advanced Server Administration Manual*.

Creating a Volume

Create a TAS volume that exports the directory in which the VOB and/or view storage are physically located. Clients use the volume name to represent the path to the physical VOB or view storage location.

NOTE: We recommend that you test the TAS installation and configuration using non-ClearCase files before attempting to use TAS to access VOBs and views. For details, see *Testing the TAS Configuration on Non-ClearCase Files* on page 56.

The procedure required to support ClearCase is summarized here:

1. Click **TAS System** in the menu frame; then click **Volumes** in the **TAS System Configuration and Administration** pane.

2. Type a name (for example, **ccstore**) in the text box.

   Ensure that the volume name is of a form that is acceptable for all realms that will access it. For example, some realms do not accept names longer than 12 characters.

   NOTE: The text box contains a symbolic name for the volume, not the pathname to the volume storage. However, it is a good idea to specify TAS volume names that correlate to the VOB and view storage paths. (For example, a TAS volume named **ccstore** may be associated

with **/ccstore** on the UNIX computer.) If these names do not correlate, examine the volume properties to determine which pathnames are associated with which volumes.

3. Click **Create**; a **New Volume Definition** pane appears. Make these changes to support ClearCase:

   - **Pathname** — Type the pathname to the virtual root of the storage area. This pathname is the root of the VOB or view storage areas for the VOB or view server. In other words, all VOB or view storage areas must be located below this pathname (but they need not be direct subdirectories of this pathname).

     For example, if you type **/ccstore**, legal VOB and view storage names for this volume are **/ccstore/vobstore**, **/ccstore/home/vobstore**, and **/ccstore/home/project/viewstore**.

   - **Volume umask** — Type **002**.

   - **Filename Case** — Select **preserve**.

   - **Support opportunistic locks** — Clear the check box.

   Click **Submit** at the bottom of the form; then click **OK** in the confirmation pane.

For details about creating and administering volumes, see *Administering Volume Attributes* in the *Volume Administration* chapter of *TotalNET Advanced Server Administration Manual*.

Configuring the File Service

To configure the TAS file service to support ClearCase:

1. Access the file service:

   a. Click **CIFS (NB) Realm** in the menu frame.

   b. Click **Manage CIFS File Services**; a list of the file services appears.

   c. Click the file service that corresponds to your TAS server; then click **Administer**. A menu of file service operations appears.

2. Click **Configuration**; an update file service form appears. Make these changes to support ClearCase:

   - **Volume references** — Select the TAS volumes this file service references and exports.

   - **Browse master** — Select **off**.

- **Umask** — Type **002**.
- **Freespace report method** — Select **root**.
- **Windows 95 logon server** — Clear this check box.
- **Windows NT logon server** — Clear this check box.

  NOTE: You cannot use the **Windows NT Logon Server** feature if the TAS volumes are to include ClearCase storage.

Click **Submit** at the bottom of the form; then click **OK** in the confirmation pane to return to the menu of file service operations.

3. Click **Authentication Options**; the **Authentication Options** form appears. Under **User-mode authentication options**, click **Local** or **Remote**.

   NOTE: You cannot use **Share mode** authentication if the TAS volumes are to include ClearCase storage.

   For assistance in determining the authentication mode for your site, see your system administrator. For details about authentication, see section 11.2, *User Authentication,* in the *TotalNET Advanced Server Administration Manual*.

4. If you select **Remote** authentication, configure the authentication as follows:

   - **Proxies**—Click **Proxies** and type the name of the proxy servers in this text box, one per line.

     NOTE: You may need to specify in the CIFS realm the IP addresses of the WINS servers for the network on which the authentication proxy server resides. (See *Enabling and Configuring the CIFS Realm* on page 51.)

   - **Use Username map** — Select this check box to ensure that the file service references the **clearcase_albd** username map specified in *Creating a TAS Username Map for clearcase_albd* on page 51.

   For details about **Remote** authentication, see section 11.2.2, *Configuring a File Service to use Proxy Authentication*, in the *TotalNET Advanced Server Administration Manual*.

   If you select **Local authentication**, configure the authentication as follows:

   - **Use Secure Passwords** — Select this check box.

NOTE: If you select **Local authentication**, you must enter every user that will access TAS file services in a local password encryption database on the server supporting those file services. If your CIFS realm contains multiple servers supporting TAS file services, you must configure a local password encryption database on each server.

- **Use Username map** — Select this check box to ensure that the file service references the **clearcase_albd** username map specified in *Creating a TAS Username Map for clearcase_albd* on page 51.

Click **Submit** at the bottom of the authentication options form. Then click **OK** in the confirmation pane to return to the menu of file service operations.

For details about **Local** authentication, see section 11.2.1, *Configuring a File Service to use Local Authentication*, in *the TotalNET Advanced Server Administration Manual*.

Start Services and Accept Service Connections

To start the TAS file services and accept service connections:

1. Click **TAS System** in the menu frame and then click **TAS System Administration**.
2. Click **Start Services** in the **TAS System Administration** pane.

   Click **OK** in the **Confirmation** pane; then click **OK** to return to the **TAS System Administration** pane.
3. In the **TAS System Administration** pane, click **Accept Service Connections**.

   Click **OK** in the **Confirmation** pane; then click **OK** to return to the **TAS System Administration** pane.

For details about starting TAS services and accepting service connections, see sections 10.1.1, *Starting TAS Services*, and 10.4.1, *Accepting Services in TAS*, in the *TotalNET Advanced Server Administration Manual*.

At this point, TAS is configured to support ClearCase. You can exit the Framework Web interface.

Configuring ClearCase to Support TAS

For all ClearCase clients on Windows NT that have the MVFS installed and will access TAS volumes, change the **MVFS Performance** settings in the ClearCase program in Control Panel as follows:

1. Click **Start**>**Settings**>**Control Panel**. Start ClearCase.
2. On the **MVFS Performance** tab:
   - Select **Override** for both **Maximum number of mnodes to keep on the free list** and **Maximum number of mnodes to keep for cleartext free list**.
   - Set the value for both to 800.
3. Click **OK** to apply the changes and close the dialog box.
4. Restart the Windows NT client.

Testing the TAS Configuration on Non-ClearCase Files

We recommend that you test the TAS installation and configuration using non-ClearCase files and directories before attempting to use TAS to provide file access to VOBs and views, as follows:

1. Create a directory structure on your TAS server (for example, **/tasstore/testdir**) and a test file in that directory (for example, **/tasstore/testdir/testfile**).
2. Install and configure TAS as described in this chapter, using **tasstore** as the volume name and **/tasstore** as the path name for the volume.
3. From a Windows NT client, create a file in the TAS volume. Then verify that the UNIX user and group settings for that file are correct.
4. Verify that all Windows NT clients can access the TAS volume, including testing permission and access restrictions, until you are confident that TAS is working properly.

Testing the TAS Configuration with ClearCase

To verify that ClearCase and TAS are working together properly:

1. On a UNIX VOB or view server, install and configure TAS as described in this chapter, creating volumes containing VOB and/or view storage.
2. Verify that your ClearCase user and group assignments are appropriate. To do so, use the tests described in the chapter on configuring ClearCase in a mixed network in *Administering ClearCase*.
3. Verify that you can access VOBs and views on the server from a UNIX client.

4. Log on to a ClearCase client on Windows NT. Use the Region Synchronizer to import VOB-tags and view-tags for VOBs and views hosted on the UNIX server into the Windows region.

5. Ensure that you can use these views and VOBs by performing some basic ClearCase operations (for example, **mkelem**, **checkin**, and **checkout**) in them.

chactivity

Changes a UCM activity

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

chact·ivity [**–c·omment** *comment* | **–cfi·le** *pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
{ [**–hea·dline** *headline activity-selector ...*] |
[**–fcs·et** *src-activity-selector* **–tcs·et** *dest-activity-selector version-pname*[**,...**] }

DESCRIPTION

The **chactivity** command modifies one or more UCM activities. Use this command for these tasks:

- Change an activity's headline
- Move versions from the change set of one activity to the change set of another activity

Note that changing the headline for an activity does not affect its name (its unique identifier). See **rename** for related information.

The destination activity must exist before you can move a change set and both the source and destination activities must be in the same stream. Use **lsactivity –long** to list the pathnames of change set versions associated with an activity.

RESTRICTIONS

Identities: No special identity required.

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

MODIFY AN ACTIVITY'S HEADLINE. *Default*: None.

–hea·dline *headline*
Specifies a new headline for the activity. The *headline* argument can be a character string of any length. Use double quotes to enclose a headline with spaces or special characters.

SPECIFYING THE ACTIVITY. *Default*: None.

activity-selector ...
Specifies one or more activities to modify.

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.

SPECIFYING THE SOURCE AND DESTINATION ACTIVITIES. *Default*: None.

–fcs·et *src-activity-selector*
Specifies the activity from which to move versions.

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.

–tcs·et *dest-activity-selector*
Specifies the activity to move versions to. These versions are recorded in the activity's change 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.

version-pname[**,...**]
One or more version-extended pathnames that specify the versions to be moved to another change set.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Change an activity's headline.

cmd-context **chactivity -headline "Fix front matter" fix_copyright**
```
Changed activity "fix_copyright".
```

- Move a version from one activity's change set to another activity's change set.

  cmd-context **chactivity -fcset update_date **
  -tcsets fix_copyright add_proc@@/main/chris_webo_dev/1

  ```
  Moved version "add_proc@@/main/chris_webo_dev/1" from activity
  "update_date" to activity "fix_copyright".
  ```

SEE ALSO

lsactivity, **mkactivity**, **rename**, **rmactivity**

chactivity

chbl

Changes a UCM baseline

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

chbl [–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment]**
{ [**–inc·remental** | **–fu·ll**] [**–level** *promotion-level*] }
baseline-selector ...

DESCRIPTION

The **chbl** command modifies one or more UCM baselines. You can modify a baseline's labeling status or assign a new promotion level to a baseline.

Baseline Labels

Baselines can be unlabeled, incrementally labeled, or fully labeled. Only labeled baselines can be used to configure streams (see the reference pages for **rebase** and **mkstream**).

Promotion Levels

Promotion levels must be defined in the baseline's project VOB, before they can applied to baselines. See the **setplevel** reference page for information on promotion levels.

The promotion levels available in a VOB can be listed by running the **describe** command on the UCM project VOB object.

RESTRICTIONS

Identities: No special identity required.

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 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.

CHANGING A BASELINE'S LABELING STATUS. *Default*: None.

–inc·remental
Changes the labeling status for an unlabeled baseline to incremental. This option has no effect if the baseline is already incrementally or fully labeled.

–fu·ll
Changes the labeling status for a baseline from unlabeled or incremental to full. This option has no effect if the baseline is already fully labeled. A **chbl –full** operation make take a long time for components with many elements.

ASSIGNING PROMOTION LEVELS. *Default*: No change in promotion level.

–level *promotion-level*
Sets the promotion level for the specified baselines. The specified promotion level must defined in the baseline's project VOB.

SPECIFYING THE BASELINE. *Default*: None.

baseline-selector ...
Specifies one or more baselines to modify.

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

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Change an unlabeled baseline to be incrementally labeled. The baseline specifier includes a VOB component, which must be the baseline's project VOB.

cmd-context **chbl -incremental testbl.121@/vobs/core_projects**

```
Begin incrementally labeling baseline "testbl.121".
Done incrementally labeling baseline "testbl.121".
```

- Change a baseline's promotion level and check the labeling status. The baseline specifier includes a VOB component, which must be the baseline's project VOB.

  cmd-context **chbl -full -level TESTED testbl.121@\vobs\core_projects**

  ```
  Change baseline "testbl.121".
  Baseline "testbl.121" is already fully labeled.
  ```

SEE ALSO

describe, **diffbl**, **lsbl**, **lscomp**, **mkbl**, **rmbl**, **setplevel**

chbl

chfolder

Modifies a UCM folder

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

chfolder [–c·omment *comment* | **–cfi·le** *comment-file-pname* |
–cq·uery | **–cqe·ach** |**–nc·omment]**
{ **[–tit·le** *title* **]** **[–to** *to-folder-selector* **]** }
folder-selector ...

DESCRIPTION

The **chfolder** command modifies one or more UCM folders. Use it for these tasks:

- To change the title of a folder
- To move a folder to another location in the folder hierarchy of a project VOB. The **RootFolder** cannot be moved.

Note that changing a folder's title does not affect its name (its unique identifier). See **rename** for related information.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if one or more of these objects are locked: the folder, the UCM project VOB.

Mastership: (Replicated VOBs only) Your 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 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.

ASSIGNING A NEW TITLE. *Default*: None.

–tit·le *title*
Specifies the new title for the folder. The *title* argument can be a character string of any length. Use double quotes to enclose a title with special characters.

MOVING A FOLDER. *Default*: None.

–to *to-folder-selector*
Specifies the new parent folder. The to-folder and the folder you are moving must belong to the same UCM project VOB.

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

SPECIFYING THE FOLDER TO CHANGE. *Default*: None.

folder-selector ...
Specifies one or more folders to modify. **RootFolder** cannot be moved.

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

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Assign a new title to the Parsers folder. Note the VOB component of the *folder-specifier* must be the folder's project VOB.

cmd-context **chfolder -title "Team Parser Projects" Parsers@/vobs/core_projects**
```
Changed folder "Parsers@/vobs/core_projects".
```

- Make the folder Core_Parsers a subfolder of **RootFolder**. Note that the folder's project VOB is given as the VOB component of the *folder-specifier*.

  cmd-context **chfolder -to RootFolder Core_Parsers@/vobs/core_projects**
  ```
  Changed folder "Core_Parsers@\vobs\core_projects".
  ```

SEE ALSO

lsfolder, mkfolder, rmfolder, rename

chfolder

chproject

Modifies a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

chproj·ect [**–c·omment** *comment* | **–cfi·le** *pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
{ [**–tit·le** *title*]
[**–amo·dcomp** *component-selector*[,...]]
[**–to** *to-folder-selector*]
[**–reb·ase_level** *promotion-level*]
[**–policy** *policy-keyword*[,...]] [**–npolicy** *policy-keyword*[,...]]
[**–crm·enable** *ClearQuest-user-database-name* | **–ncr·menable**] }
project-selector ...

DESCRIPTION

The **chproject** command modifies one or more UCM projects. Use it to:

- Change a project's title
- Add one or more modifiable components to a project
- Move a project to another folder
- Change the promotion level required of a baseline before it can be used in a rebase operation.
- Set policy for a project.
- Enable or disable a project for use with Rational ClearQuest

Project Titles

Note that changing a project's title does not affect its name (its unique identifier). See **rename** for related information.

Adding New Components

Over time, a project's scope can broaden, and you may need to add writable components to the project's integration stream. The **–amodcomp** option allows you to add one or more modifiable components. Components can be added to project development streams with the **rebase –baseline** command.

Setting Required Promotion Levels for Recommended Baselines

A project's rebase level is defined as the minimum promotion level a baseline must have to be recommended in a rebase operation. For example, if ProjectA has three promotion levels, **REJECTED**, **TESTED**, and **RELEASED** (in ascending order), and **TESTED** in the rebase level, only baselines that are labeled **TESTED** or **RELEASED** are included in the project's list of recommended baselines. See **rebase** and **setplevel** for more information.

Project Policies

You can set or unset projectwide policies, such as specifying that views attached to the integration stream must be snapshot views. Policies are identified on the command line by their keyword. The following table describes these policies and lists the keywords used to set them.

Policy	Keyword
Recommend snapshot views for integration work. Dynamic views are suggested if this policy is not set.	**POLICY_UNIX_INT_SNAP** (UNIX) or **POLICY_WIN_INT_SNAP** (Windows)
Recommend snapshot views for development work. Dynamic views are suggested if this policy is not set.	**POLICY_UNIX_DEV_SNAP** (UNIX) or **POLICY_WIN_DEV_SNAP** (Windows)
Require a development stream to be based on the current recommended baselines before it can be used to deliver changes to the integration stream.	**POLICY_DELIVER_REQUIRE_REBASE**
Do not allow delivery from a development stream that has checkouts.	**POLICY_DELIVER_NCO_DEVSTR**

Using Rational ClearQuest with UCM projects

You can link or unlink a UCM project to a ClearQuest database with the **–crmenable** or **–ncrmenable** options. When you ClearQuest-enable a UCM project that contains UCM activities, for each UCM activity, a ClearQuest record of type UCMUtilityActivity is created and linked to

the activity. This process is called *activity migration*. If you disable a link to ClearQuest, from a UCM project that contains activities, all its activities are unlinked from their ClearQuest records.

All ClearQuest-enabled projects in the same UCM project VOB must link to the same ClearQuest user database.

The –**crmenable** and –**ncrmenable** options display a summary of the number of activities that have been migrated or unlinked.

You are informed if activities cannot be migrated or linked because they are not mastered in the current UCM project VOB replica. If any are discovered, you are informed of the number of activities for which this is true and shown a list of replicas from which to run the command again to correct the problem.

Detecting and Correcting Incorrectly Enabled Activities

You can also use the –**crmenable** and –**ncrmenable** options to check for possible linking errors. If you believe that your ClearQuest-enabled project may contain activities that are not linked to a ClearQuest record, run the **chproject** –**crmenable** command. This scans all activities in the project, skipping activities that are already linked and migrating all activities that are not linked. To check for linked activities in projects that have been disabled for use with ClearQuest, run the **chproject** –**ncrmenable**. This removes links between activities as needed. See *Managing Software Projects with ClearCase* for further information.

RESTRICTIONS

Identities: You must be the project owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—local administrator of the ClearCase LT server host

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

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: **–cq**). See 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.

ASSIGNING A NEW TITLE. *Default*: None.

–tit·le *title*
Specifies a new title for the project. The *title* argument can be a character string of any length. Enclose a title with special characters in double quotes

ADDING TO THE LIST OF MODIFIABLE COMPONENTS FOR A PROJECT. *Default*: None.

–amo·dcomp *component-selector*[,...]
Adds one or more components to the project's set of modifiable components.

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

MOVING THE PROJECT TO ANOTHER FOLDER. *Default*: None.

–to *to-folder-selector*
Moves one or more projects to the specified folder.The to-folder and project must have the same UCM project VOB.

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

CHANGING THE RECOMMENDED PROMOTION LEVEL FOR A REBASE OPERATION. *Default*: None.

–reb·ase_level *promotion-level*
Changes the promotion level required for baselines to be recommended baselines in a rebase operation. For each component, the latest baseline in the integration stream at or above this promotion level is recommended.

SETTING PROJECT POLICY. *Default*: None.

–policy *policy-keyword*
Activates the specified policy. See *Project Policies* on page 72

–npolicy *policy-keyword*
Removes the specified policy. See *Project Policies* on page 72

LINKING A PROJECT TO RATIONAL CLEARQUEST. *Default*: None.

–crm·enable *ClearQuest-user-database-name*
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.

–ncr·menable
Disables use of Rational ClearQuest.

SELECTING A PROJECT. *Default*: None.

project-selector ...
Specifies one or more projects to modify.

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

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Add the modifiable component, **webo_modeler**, to the project.

cmd-context **chproject -amod webo_modeler webo_proj1@/vobs/webo_pvob**

```
Changed modifiable component list for project
"webo_proj1@/vobs/webo_pvob".
```

SEE ALSO

chbl, **lscomp**, **lsproject**, **mkproject**, **mkcomp**, **rebase**, **rmproject**

chproject

chstream

Modifies a UCM stream

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

chstream [**–c·omment** *comment* | **–cfi·le** *pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–title** *title*] *stream-selector ...*

DESCRIPTION

The **chstream** command allows you to assign a new title to a stream. The stream's UUID (universal unique identifier) is not changed. See **rename** for related information.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if there are locks on the following objects: the UCM project VOB, 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 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 NEW STREAM TITLE *Default:* None.

–title *title*

Specifies the new title for the stream. The *title* argument can be a character string of any length. Enclose a title with special characters in double quotes.

SPECIFYING THE STREAM. *Default:* None.

stream-selector ...

Specifies one or more streams to be modified.

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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Change the title of a stream.

  cmd-context `chstream -title "jamaica blue" java-int@vobs/javaprojvob`

SEE ALSO

lsstream, **mkstream**, **rename**, **rmstream**

cleardiffbl

Starts the **diffbl** browser

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

Platform
UNIX
Windows

SYNOPSIS

cleardiffbl [*baseline-selector1 baseline-selector2*]

DESCRIPTION

The **cleardiffbl** command invokes a graphical version of the **diffbl** utility, which compares two baselines and displays differences in terms of activities or versions.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

baseline-selector1

baseline-selector2
Specifies the two baselines to compare. *baseline-selector* is of the form: [**baseline:**]*baseline-name*[*@vob-selector*] and *vob* is the baseline's UCM project VOB.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Display the differences between the baselines **rev17bl1** and **rev17bl2.**

  cmd-context **cleardiffbl rev17bl1 rev17bl2**

SEE ALSO

diffbl

clearfsimport

Converts file system objects to element versions

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

Platform
UNIX
Windows

SYNOPSIS

- UNIX only:

  clearfsimport [**–preview**] [**–follow**] [**–recurse**] [**–rmname**] [**–comment** *comment*] [**–mklabel** *label*] [**–nsetevent**] [**–identical**] [**–master**] [**–unco**] *source-name* [. . .] *target-VOB-directory*

- Windows only:

  clearfsimport [**–preview**] [**–recurse**] [**–rmname**] [**–comment** *comment*] [**–mklabel** *label*] [**–nsetevent**] [**–identical**] [**–master**] [**–unco**] [**–downcase**] *source-name* [. . .] *target-VOB-directory*

DESCRIPTION

The **clearfsimport** command reads the specified file system source objects and places them in the target VOB. This command uses magic files to determine which element type to use for each element created (see the **cc.magic** reference page).

RESTRICTIONS

Identities: You must be **root** (UNIX) or the VOB owner to run **clearfsimport** unless you invoke it with the **-nsetevent** option.

Locks: If it encounters a VOB lock while trying to write data during an import operation, **clearfsimport** will pause and retry the operation every 60 seconds until it succeeds.

OPTIONS AND ARGUMENTS

PREVIEWING THE RESULTS. *Default:* No preview.

–preview
Previews the import, listing all elements that the import would add or change, as well as any checkouts that would conflict with imports, but does not import anything.

HANDLING OF UNIX SYMBOLIC LINKS. *Default:* Processes each UNIX symbolic link as a VOB symbolic link with the same link text.

–follow
Processes the object to which a UNIX symbolic link points, instead of importing the link itself into the VOB.

HANDLING OF DIRECTORY ARGUMENTS. *Default:* If a *source-name* argument names a directory to be imported, an element version is created for the directory and for each file, directory, or UNIX symbolic link residing at the top level of the directory.

–recurse
Descends recursively into all *source-name* arguments that are directories.

HANDLING OF EXISTING VOB DIRECTORIES. Default: Existing VOB directories that are not present in the sources to be imported are left as is.

–rmname
For all *source-name*s that are directories, performs an **rmname** operation on elements that already existed in the VOB but are not present in the source directory. If used in combination with **-recurse**, performs this **rmname** operation in all directories traversed.

COMMENTS. *Default:* **created by clearfsimport**

–comment *comment*
Attaches the specified comment instead of the default comment to each element version checked in to the VOB.

LABELING. *Default:* No labeling.

–mklabel *label*
Attaches the specified label instance to each element version checked in. If the corresponding label type does not exist, it is created. If the label is already attached to an existing element version, it is moved.

EVENT RECORDS. *Default:* Historical information associated with the sources is preserved.

–nsetevent
Specifies that event records and historical information for new elements and element versions show the user who executed **clearfsimport** and the date of execution, not the original data associated with the sources. This option creates element versions that are

newer than the original sources; thus, the **clearfsimport** operation is not restartable after you have invoked it with this option.

CREATION OF IDENTICAL SUCCESSOR VERSIONS. *Default:* Element versions that are identical to their predecessors in the source are not created.

–identical
Creates a new version of an element—even if it is identical to its predecessor—if the source has a more recent date than that of the version in the VOB.

BRANCH MASTERSHIP. *Default:* The main branch of the element is mastered by the replica that masters the branch's type.

–master
Assigns mastership of the main branch of the element to the VOB replica at which you execute this command.

HANDLING OF EXISTING CHECKED OUT ELEMENTS. *Default:* When **clearfsimport** encounters a checked out element that already exists in the target VOB and that corresponds to a source to be imported, it prints an error and continues.

–unco
If a checked-out file element corresponding to a source file to be imported already exists in the target VOB, an **uncheckout** operation is executed on the element and the corresponding view-private file is retained with a suffix of **.keep**. The import operation then proceeds to check the element out again, then checks in the imported version.

HANDLING OF CASE OF IMPORTED ELEMENTS. *Default:* Case preserving.

–downcase
Forces downcasing of imported elements.

NOTE: When importing files from a UNIX host into a VOB on a Windows NT host, **clearfsimport** may be unable to operate correctly on files and directories that have mixed-case names. For example, if a mixed-case name is specified on the command line, **clearfsimport** will create the element name as specified, even if the case mix of the source is different. In addition, the **-rmname** option will not work where source and target elements differ only in character case. Consistent use of **-downcase** can help avoid these problems. If **-downcase** is specified for an initial import from UNIX to Windows NT, it should be specified on any subsequent imports into the same VOB directories.

SPECIFYING THE SOURCE. *Default:* None.

source-name [. . .]
Flat files, directories, and UNIX symbolic links to be imported to the VOB.

For each pathname, the leaf of the pathname is imported into the target VOB. For example, **/usr/src/lib/foo.c** is imported into the VOB **/vobs/mylib** as **/vobs/mylib/foo.c** or into the VOB **/bigvob** as **/bigvob/foo.c**.

For each file, an element version is imported. If a corresponding version of the file already exists in the target VOB and is checked out, **clearfsimport** prints and error and continues unless the **-unco** option was specified. A summary of import failures due to checked out elements is printed at the completion of the import operation.

For each directory, versions are created for all files and UNIX symbolic links it contains. Also created is a directory element with one version for the directory and each of its subdirectories. Checked out VOB directories corresponding to source directories to be imported are re-used.

SPECIFYING THE TARGET VOB. *Default:* None.

target-VOB-directory
The VOB directory to which the sources are to be imported.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Preview a VOB—**/vobs/projectx/src**—that is to be populated with the contents of **/usr/src/projectx**. Recursively descend all directories encountered and follow UNIX symbolic links to the target objects.

  cmd-context **clearfsimport –preview –follow –recurse /usr/src/projectx /vobs/projectx/src**

SEE ALSO

cc.magic, **events_ccase**, **relocate**, **rmname**, **uncheckout**

clearjoinproj

Starts the UCM Join Project Wizard

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

Platform
UNIX
Windows

SYNOPSIS

clearjoinproj

DESCRIPTION

The **clearjoinproj** command starts the UCM Join Project Wizard, which takes you through the steps required to start work on an existing UCM project.

You can also start the Join Project Wizard from the Project Explorer.

RESTRICTIONS

Identities: No special identity required.

Locks: A stream cannot be created if there are locks on any of the following objects: the project VOB and, for integration streams, the project.

Mastership: There are no mastership requirements.

OPTIONS AND ARGUMENTS

None.

EXAMPLE

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Invoke the Join Project wizard.

  cmd-context **clearjoinproj**

SEE ALSO

clearprojexp, **mkstream**, **mkview**

clearmake

ClearCase build utility; maintains, updates, and regenerates groups of programs

APPLICABILITY

Product	Command Type
ClearCase	command

Platform
UNIX
Windows

SYNOPSIS

- UNIX only—Build a target:

  clearmake [**–f** *makefile*] ... [**–cukinservwdpqUNR**]
  [**–J** *num*] [**–B** *bldhost-file*] [**–C** *compat-mode*] [**–V** | **–M**] [**–O** | **–T** | **–F**]
  [**–A** *BOS-file*] ... [*macro***=***value* ...] [*target-name* ...]

- Windows only—Build a target:

  clearmake [**–f** *makefile*] ... [**–cukinservwdpqUNR**]
  [**–C** *compat-mode*] [**–V** | **–M**] [**–O** | **–T** | **–F**]
  [**–A** *BOS-file*] ... [*macro***=***value* ...] [*target-name* ...]

- Display version information for **clearmake**:

  clearmake { **–ver**·**sion** | **–VerAll** }

DESCRIPTION

clearmake is ClearCase's variant of the UNIX **make(1)** utility. It includes most of the features of UNIX System V **make(1)**. It also features compatibility modes, which enable you to use **clearmake** with makefiles that were constructed for use with other popular **make** variants, including **Gnu make**.

clearmake features a number of ClearCase extensions:

- **Configuration Lookup**—A build-avoidance scheme that is more sophisticated than the standard scheme, which uses time stamps of built objects. Configuration lookup also includes automatic dependency detection. For example, this guarantees correct build

behavior as C-language header files change, even if the header files are not listed as dependencies in the makefile.

- **Derived Object Sharing**—Developers working in different views can share the files created by **clearmake** builds.
- **Creation of Configuration Records**—Software bill-of-materials records that fully document a build and support the ability to rebuild.

NOTE: **clearmake** is intended for use in dynamic views. You can use **clearmake** in a snapshot view, but most of the features that distinguish it from ordinary **make** programs—build avoidance, build auditing, derived object sharing, and so on—are not enabled in snapshot views. (Parallel builds are enabled.) The rest of the information in this reference page assumes you are using **clearmake** in a dynamic view.

Related Reference Pages

The following reference pages include information related to **clearmake** operations and results:

abe	Executes builds on remote hosts during a parallel build.
bldhost	Specifies hosts to be used for parallel builds.
bldserver.control	Controls use of a host for parallel builds.
omake	Builds software on Windows NT and provides compatibility with PC-based **make** products.
clearaudit	Runs audited builds.
lsdo	(**cleartool** subcommand) Lists derived objects created by **clearmake**, **omake**, or **clearaudit**.
catcr	(**cleartool** subcommand) Displays configuration records created by **clearmake**, **omake**, or **clearaudit**.
diffcr	(**cleartool** subcommand) Compares configuration records created by **clearmake**, **omake**, or **clearaudit**.
rmdo	(**cleartool** subcommand) Removes a derived object from a VOB.
winkin	(**cleartool** subcommand) Winks in a derived object to a view or to the VOB.

See also *Building Software with ClearCase*.

View Context Required

For a build that uses the data in one or more VOBs, the shell or command interpreter from which you invoke **clearmake** must have a view context:

- On UNIX systems, the view context must be either a set view or a working directory view. If you have a working directory view, but it differs from the set view, **clearmake** changes its set view to the working directory view.

- On Windows systems, you must be on the dynamic-views drive (default: **M:**) or a drive assigned to a view. If you want derived objects to be shared among views, you must be on a drive assigned to a view.

You can build objects in a standard directory, without a view context, but this disables many of **clearmake**'s special features.

clearmake AND MAKEFILES

clearmake is designed to read makefiles in a way that is compatible with other **make** variants. For details, including discussions of areas in which the compatibility is not absolute, see *Using clearmake Compatibility Modes* in *Building Software with ClearCase*.

For more information about makefiles and **clearmake**, see *Building Software with ClearCase*.

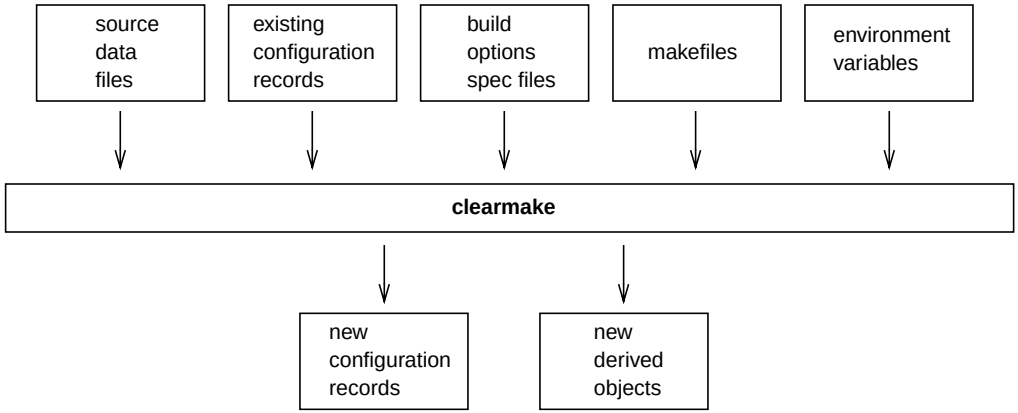
HOW BUILDS WORK

In many ways, ClearCase builds adhere closely to the standard **make** paradigm:

1. You invoke **clearmake**, optionally specifying the names of one or more targets. (Such explicitly specified targets are called "goal targets.")
2. **clearmake** reads zero or more makefiles, each of which contains targets and their associated build scripts. It also reads zero or more build options specification (BOS) files, which supplement the information in the makefiles.
3. **clearmake** supplements the makefile-based software build instructions with its own built-in rules, or, when it runs in a compatibility mode, with built-in rules specific to that mode.
4. For each target, **clearmake** performs build avoidance, determining whether it actually needs to execute the associated build script (target rebuild). It takes into account both source dependencies (Have any changes occurred in source files used in building the target?) and build dependencies (Must other targets be updated before this one?).
5. If it decides to rebuild the target, **clearmake** executes its build script.

The following sections describe special **clearmake** build features in more detail. Figure 7 illustrates the associated data flow.

Figure 7 Data Flow in a clearmake Build



CONFIGURATION RECORDS AND DERIVED OBJECTS

In conjunction with the MVFS file system, **clearmake** audits the execution of all build scripts, keeping track of file use at the system-call level. For each execution of a build script, it creates a configuration record (CR), which includes the versions of files and directories used in the build, the build script, build options, (for example, macro assignments) and other related information. A copy of the CR is stored in the VOB database of each VOB in which the build script has built new objects.

A file created within a VOB by a build script is called a derived object (DO), and it can be shareable or nonshareable. When a shareable derived object is built in a view, a corresponding VOB database object is also created. This enables any view to access and possibly share (subject to access permissions) any derived object, no matter what view it was originally created in.

When a build tool creates a nonshareable derived object, the tool does not write any information about the DO to the VOB. Therefore, the DO is invisible to other views and cannot be winked in by them. Builds that create nonshareable DOs are called express builds. For more information about using express builds, see *Preventing Winkin to Other Views* on page 92.

NOTE: Symbolic links created by a build script and files created in non-VOB directories are not DOs. See *MVFS FILES AND NON-MVFS OBJECTS* on page 92.

For each build script execution, ClearCase logically associates each DO created with the build script's CR.

You can suppress the creation of CRs and derived objects with the **–F** option. For details on CRs, derived objects, see *Derived Objects and Configuration Records* in *Building Software with ClearCase*. For information on ClearCase-specific special targets, see *Building Software with ClearCase*.

Configuration Record Hierarchies

A typical makefile has a hierarchical structure. Thus, a single invocation of **clearmake** to build a high-level target can cause multiple build scripts to be executed and, accordingly, multiple CRs to be created.

CONFIGURATION LOOKUP AND WINKIN

For directory targets, **clearmake** uses standard **make** logic.

When a target names a nondirectory file in a VOB, **clearmake** (by default) uses configuration lookup to determine whether a build is required. This involves comparing the CRs of existing DOs with the current build configuration:

- The versions of elements selected by the view's config spec.
- The build options to be applied, as specified on the **clearmake** command line, in the environment, in makefile(s), or in build options specification file. See *BUILD OPTIONS SPECIFICATION FILE* on page 93.
- The build script to be executed.

In performing configuration lookup, **clearmake** considers a DO version (a derived object that has been checked in as a version of an element) only if the version's element has the same pathname as the original derived object. That is, if you copy a DO to a different location from where it was created and check it in there, **clearmake** does not consider the DO version.

clearmake first tries to avoid rebuilding by reusing a DO in the current view; this succeeds only if the CR of the candidate DO matches the current build configuration. For the purpose of rebuilding, a *branch*/0 version of a file selected by a view is considered to match its non-zero predecessor version in a CR.

clearmake can also avoid rebuilding by finding a shareable DO, built in another view, whose CR matches the current build configuration. In this case, it winks in (**winkin**) that derived object, causing it to be shared among views. Other derived objects created by the same build script (siblings) are winked in at the same time. **clearmake** rebuilds a target only if it cannot locate any existing derived object that matches the current build configuration.

DO versions must be checked out before they can be re-used or winked in. The **-c** option to **clearmake** provides support for automatically checking out these DOs before they are used. The **CCASE_AUTO_DO_CI** environment variable provides a means to automatically check in DOs checked out by **clearmake -c.** Checkouts executed by this feature behave like any **cleartool checkout** does with respect to reservation. Methods that can be used to change **cleartool checkout**'s default reservation policy apply here as well. The checkouts are not audited. Checkins preserve the timestamp of the DO as though **cleartool checkin -ptime** were used. This feature is fully compatible with **checkout** or **checkin** triggers, which fire normally when the event occurs.

The .cmake.state File

The **.cmake.state** file is a view-private cache of config records for derived objects built in the view. **clearmake** creates this file in the directory that was current when the build started. During subsequent builds in that directory in the view, **clearmake** references the file instead of communicating with the VOB. This makes configuration lookup faster, improving **clearmake** performance.

You can delete **.cmake.state** files if they get too large. When **clearmake** looks for a **.cmake.state** file and it doesn't exist, no errors occur and **clearmake** creates a new file.

Suppressing Configuration Lookup

You can override the default configuration lookup behavior with command options and ClearCase special targets. (For information on ClearCase special targets, see *Building Software with ClearCase*). For example, **–T** turns off configuration lookup, basing rebuild decisions on time stamps, and **–V** disables winkin of DOs from other views.

Preventing Winkin to Other Views

You can prevent derived objects that you create from being winked in to other views by using express builds. During an express build, **clearmake** creates nonshareable DOs. These DOs have config records, but **clearmake** does not write information about the DOs into the VOB. DOs created during an express build are invisible to other views. To use express builds, invoke **clearmake** in a view configured with the nonshareable DOs property:

- To configure an existing view for express builds, use **chview –nshareable_dos** *view-tag*. See the **chview** reference page for more information.
- To create a new view and configure it with the nonshareable DOs property, use the **mkview** command and specify the **–nshareable_dos** option. See the **mkview** reference page for more information.
- Use the **–T** or **–F** options to create view-private files only.
- Use special targets that prevent winkin; for example, **.NO_WINK_IN**. For more information, see *Building Software with ClearCase*.
- Set an environment variable that no other users set identically. This environment variable will be recorded in the config records **clearmake** creates.

MVFS FILES AND NON-MVFS OBJECTS

All files with pathnames below a VOB-tag (VOB mount point) are termed MVFS files:

- Checked-in versions of file elements (data stored in VOB)
- Checked-out versions of file elements (data stored in view)
- Other view-private files
- Derived objects

Conversely, a non-MVFS object is any file, directory, or UNIX link whose pathname is not under a VOB-tag; such objects are not version controlled. By default, non-MVFS objects are not audited during **clearmake** builds. Non-MVFS files that are read during a build are not included in the detected dependency list of the CR, and non-MVFS files that are created are not ClearCase derived objects. A CR includes information on a non-MVFS object used by a build script only if either of these conditions are true:

- The object appears as an explicit dependency in the makefile.
- The object can be inferred to be a dependency through **clearmake**'s file-name extension rules.

The explicit dependency is referred to as a makefile dependency. For example:

```
src.o : /usr/include/stdio.h
```

UNIX Systems Only—Non-MVFS Files in Configuration Lookup and Remote Building

During configuration lookup, **clearmake** examines each non-MVFS file that is listed in the CR of a candidate DO. The CR entry includes: the non-MVFS file's size, its time stamp, and its checksum. The current version of the non-MVFS file must match the CR entry in one of these ways:

- First check: file size and time stamp
- Second check: file size and checksum

clearmake also performs these checks during a parallel build. If the characteristics of a file are different on the local machine and the remote build host, **clearmake** does not attempt the rebuild; instead, it prints the following message:

```
abe: Error: Inconsistent version for dependency "dependency"
```

This ensures the consistency of a build across multiple hosts.

BUILD OPTIONS SPECIFICATION FILE

A build options specification (BOS) file is a text file containing macro definitions and/or ClearCase special targets. We recommend that you place nonpermanent option specifications (for example, a macro that specifies "compile for debugging") in a BOS file, instead of on the **clearmake** command line. This minimizes the likelihood of having **clearmake** perform a rebuild unexpectedly (for example, because you specified **–g** (UNIX) or **/Zi** (Windows) on a compiler command line last time, but forgot to specify it this time).

See *Building Software with ClearCase* for details.

clearmake SLEEP

clearmake can monitor the current VOB's lock status during a build, so that if an administrator locks the VOB while **clearmake** is running, the build does not terminate abnormally. Before executing the build script and before creating a derived object and configuration record,

clearmake checks the lock status of the current VOB. If the VOB is locked, **clearmake** starts a sleep-check cycle. When it finds the VOB unlocked, the build proceeds.

NOTE: **clearmake** starts the sleep-check cycle even if the user who invokes the build is on the exception list for the lock.

When a sleep-check cycle begins, **clearmake** prints a message announcing the sleep, its duration, and the reason for it. Initially, **clearmake** checks the lock status 10 times, waiting 60 seconds between attempts. **clearmake** then increments the sleep time by 5 seconds and again tries 10 times, and so on. **clearmake** prints a sleep message at the start of each group of 10 retries.

This implementation does not guarantee that the build will not terminate abnormally. There are still a few "windows of failure." The build script will fail and terminate abnormally, and the build will terminate if any of these conditions is true:

- The build script modifies the VOB, either by running a **cleartool** command that modifies the VOB, or simply removing the derived object which is the target of the build.
- The build script writes to another VOB other than the current VOB, and the other VOB is locked.
- The VOB becomes locked in the short time between the check and the build script execution, and the build script has an action that modifies the VOB.

By default, **clearmake** checks the VOB containing the working directory that was current at the start of the build. To check a set of VOBs, set the environment variable **CCASE_BLD_VOBS** to the list of VOB-tags to check. Separate the VOB-tags in the list with a space, tab, colon (UNIX), semicolon (Windows), or comma.

To disable the checks, set the environment variable **CCASE_BLD_NOWAIT**. When this environment variable is set, **clearmake** does not check for a VOB-lock (or wait for the VOB to be unlocked).

CACHING UNAVAILABLE VIEWS

When **clearmake** shops for a derived object to wink in to a build, it may find DOs from a view that is unavailable (because the view server host is down, the **albd_server** is not running on the server host, and so on). Attempting to fetch the DO's configuration record from an unavailable view causes a long time-out, and the build may reference multiple DOs from the same view.

clearmake and other **cleartool** commands that access configuration records and DOs (**lsdo**, **describe**, **catcr**, **diffcr**) maintain a cache of tags of inaccessible views. For each view-tag, the command records the time of the first unsuccessful contact. Before trying to access a view, the command checks the cache. If the view's tag is not listed in the cache, the command tries to contact the view. If the view's tag is listed in the cache, the command compares the time elapsed since the last attempt with the time-out period specified by the **CCASE_DNVW_RETRY** environment variable. If the elapsed time is greater than the time-out period, the command removes the view-tag from the cache and tries to contact the view again.

NOTE: The cache is not persistent across **clearmake** sessions. Each recursive or individual invocation of **clearmake** attempts to contact a view whose tag may have been cached in a previous invocation.

The default time-out period is 60 minutes. To specify a different time-out period, set CCASE_DNVW_RETRY to another integer value (representing minutes). To disable the cache, set CCASE_DNVW_RETRY to 0.

UNIX ONLY—PARALLEL BUILDING

clearmake supports parallel building (execution of several build scripts concurrently on one or more hosts). Parallel building is enabled by the **–J** option, which specifies the parallelism (concurrency) level, and the build hosts file, which lists hosts where build scripts can be dispatched.

Before starting a parallel build, **clearmake** determines what work needs to be done, organizing the work as a sequence of target rebuilds. **clearmake** then dispatches build scripts to hosts, using a load balancing scheme. By default, a host is used only if it is at least 50% idle. You can adjust this idleness threshold with a **–idle** specification in your build hosts file. See the **bldhost** and **bldserver.control** reference pages for details.

To suppress parallel building for some or all of a makefile's targets, use the special **.NOTPARALLEL** target. See *Building Software with ClearCase* for details.

For information on setting up a parallel build, see *Setting Up a Parallel Build* in *Building Software with ClearCase*.

UNIX Only—Remote Build Environment

clearmake dispatches a build script to a remote host by invoking a remote shell there. This shell, in turn, runs an audited build executor (**abe**) process which executes the build script. If the **abe** process cannot be started, **clearmake** will not use the host.

For information on how the environment is set, see the **abe** reference page.

UNIX Only—Terminal Output

In a serial build (**–J** not specified), a target's build script is connected to **stdout** directly. Output appears as soon as it is produced by the script's commands. In a parallel build (**–J** specified with an argument >=1), the standard output of each build script is accumulated in a temporary file by **clearmake**. As each build script finishes, **clearmake** sends it to **stdout** all at once.

UNIX Only—Enabling Parallel Building on the Local Host

To perform a parallel build on the local host, make sure that both of the following conditions are true:

- CCASE_HOST_TYPE is unset when you invoke **clearmake**.
- You specify **–J** *num* on the **clearmake** command line, where *num* is greater than 0.

clearmake uses the idle specification in your host's **bldserver.control** file to determine whether it can perform the build on your host. If your host does not have a **bldserver.control** file, **clearmake** assumes an idle threshold of 0 and performs the build regardless of the load on your host.

NOTE: **clearmake** prints a message that it is performing a parallel build on the local host.

UNIX Only—Parallel Build Scheduler

clearmake schedules and manages target rebuilds as follows:

- It executes the build script for an out-of-date target as soon after detection as system build resources will allow.
- It does not assume that executing a build script for a specific target implies that the target was updated.

clearmake evaluates the dependency graph, beginning with the command-line supplied targets. Before evaluating a specific target, **clearmake** ensures that all dependents of that target have been evaluated and brought up to date. As soon as a target is deemed to be out of date, it is made available for rebuilding. A rebuild is initiated as soon as system resources allow. Depending on the availability of build hosts and load-balancing settings, this may happen immediately or be delayed.

When DO shopping/winkin occurs, **clearmake** postpones DO lookup for any target that has scheduled dependents until the target is encountered in the rebuild logic. When a target with previously scheduled dependents is encountered in the rebuild logic, **clearmake** then performs the DO shopping/winkin attempt only when the target's dependencies have completed. This eliminates unnecessary rebuilds in serial mode and allows a parallel **clearmake** to initiate rebuilds sooner.

UNIX Only—Building Targets on Specified Hosts

When you perform a parallel build with **clearmake**, you can specify that **clearmake** must build a target on a certain host. The environment variable **CCASE_BLD_HOSTS** specifies one or more build hosts.

NOTE: These hosts do not have to appear in your build hosts file. If a specified host appears in your build hosts file, **clearmake** ignores any **–idle** specifications for the host in the build hosts file and uses **–idle 0**.

We recommend that you set this variable conditionally in your makefile, using target-dependent variable bindings. If you set the variable on the **clearmake** command line, in your process environment, or unconditionally in your makefile, it applies to all targets.

NOTE: **clearmake** supports target-dependent variable bindings in standard mode and in Sun compatibility mode. You can also use target-dependent variable bindings in your BOS file for any compatibility mode.

For example, to ensure that the target **foo** is built on host **neon** or **saturn**:

```
foo := CCASE_BLD_HOSTS = neon saturn
```

You can also use patterns in target names. For example, to build all **.o** files on host **pluto**:

```
%.o := CCASE_BLD_HOSTS = pluto
```

clearmake applies **CCASE_BLD_HOSTS** bindings to dependencies of the specified targets. To apply **CCASE_BLD_HOSTS** to the specified targets but not their dependencies, add the line shown below to the builtins file for your compatibility mode:

Mode	Location of builtins file	Line to add
standard	*ccase-home-dir*/**etc/builtin.mk**	`% := CCASE_BLD_HOSTS =`
Sun	*ccase-home-dir*/**etc/sunbuiltin.mk**	`% := CCASE_BLD_HOSTS =`

BUILD REFERENCE TIME AND BUILD SESSIONS

clearmake takes into account the fact that as your build progresses, other developers can continue to work on their files, and may check in new versions of elements that your build uses. If your build takes an hour to complete, you do not want build scripts executed early in the build to use version 6 of a header file, and scripts executed later to use version 7 or 8. To prevent such inconsistencies, **clearmake** locks out any version that meets both of these conditions:

- The version is selected by a config spec rule that includes the **LATEST** version label.
- The version was checked in after the time the build began (the build reference time).

This reference-time facility applies to checked-in versions of elements only; it does not lock out changes to checked-out versions, other view-private files, and non-MVFS objects. **clearmake** adjusts for the fact that the system clocks on different hosts in a network may be somewhat out of sync (clock skew).

For more information, see *Pointers on Using ClearCase Build Tools* in *Building Software with ClearCase*.

EXIT STATUS

clearmake returns a zero exit status if all goal targets are successfully processed. It returns a nonzero exit status in two cases:

- **clearmake** itself detects an error, such as a syntax error in the makefile. In this case, the error message includes the string “clearmake”.
- A makefile build script terminates with a nonzero exit status (for example, a compiler error).

See also the description of the **–q** option.

OPTIONS AND ARGUMENTS

clearmake supports the options below. In general, standard **make** options are lowercase characters and **clearmake** extensions are uppercase. Options that do not take arguments can be combined on the command line (for example, **–rOi**).

–f *makefile*
Use *makefile* as the input file. If you omit this option, **clearmake** looks for input files named **makefile** and **Makefile** (in that order) in the current working directory. You can use more than one **–f** *makefile* argument pair. Multiple input files are effectively concatenated.

–u
(Unconditional) Rebuild all goal targets specified on the command line, along with the recursive closure of their dependencies, regardless of whether they need to be rebuilt. (See also **–U**.)

–k
Abandon work on the current entry if it fails, but continue on other targets that do not depend on that entry.

–i
Ignore error codes returned by commands.

–n
(No-execute) List command lines from the makefile for targets which need to be rebuilt, but do not execute them. Even lines beginning with an at-sign (@) are listed. See *Building Software with ClearCase*.

Exception: A command containing the string **$(MAKE)** is always executed on Windows systems. On UNIX systems, it is executed unless you are using **sgismake** or **sgipmake** compatibility mode. These modes do not necessarily execute **$(MAKE)**.).

–s
(Silent) Do not list command lines before executing them.

–e
Environment variables override macro assignments within the makefile. (But *macro=value* assignments on the command line or in a build options spec override environment variables.)

–r
(No-rules) Do not use the built-in rules in file *ccase-home-dir***/etc/builtin.mk** (UNIX) or *ccase-home-dir***\etc\builtin.mk** (Windows). When used with **–C**, this option also disables reading platform-specific startup files. See the **–C** option for more information.

–v

(Verbose) Slightly more verbose than the default output mode. These features are particularly useful:

- Listing of why **clearmake** does not reuse a DO that already appears in your view (for example, because its CR does not match your build configuration, or because your view does not have a DO at that pathname)
- Listing of the names of DOs being created

–w

(Working directory) Prints a message containing the working directory both before and after executing the makefile.

–d

(Debug) Quite verbose; appropriate only for debugging makefiles.

–p

(Print) Lists all target descriptions and all macro definitions, including target-specific macro definitions and implicit rules, and stops before executing anything.

–q

(Query) Evaluates makefile targets, but does not run the build scripts. **clearmake** returns 0 if the targets are up to date, and 1 if any targets need to be rebuilt. Note that **clearmake** treats a winkin of a derived object as a rebuild, so **clearmake –q** returns 1 if a DO can be winked in for a target.

–c

(Check out DOs) Before building or winking in a target, **clearmake** determines whether the target is a checked-in DO visible in the view at the path named in the makefile. If such a DO is found, **clearmake -c** checks it out before rebuilding it or winking it in. If a target creates sibling DOs, target group syntax must be used in the makefile or siblings will not be subject to this behavior.

–U

Unconditionally builds goal targets only. Subtargets undergo build avoidance. If you don't specify any target on the command line, the default target is the goal. (The **–u** option unconditionally builds both goal targets and build dependencies.)

–N

Disables the default procedure for reading one or more BOS files. For a description of the default procedure, see *Building Software with ClearCase*.

–R

(Reuse) Examines sibling derived objects (objects created by the same build rule that created the target) when determining whether a target object in a VOB can be reused (is up to date). By default, when determining whether a target can be reused, **clearmake**

ignores modifications to sibling derived objects. **–R** directs **clearmake** to consider a target out of date if its siblings have been modified or deleted.

–J *num*

Enables **clearmake**'s parallel building capability. The maximum number of concurrent target rebuilds is set to the integer *num*. If *num*=0, parallel building is disabled. (This is equivalent to not specifying a **–J** option at all.)

Alternatively, you can specify *num* as the value of environment variable **CCASE_CONC**, described in *Special Environment Variables* on page 105.

For more information, see *UNIX ONLY—PARALLEL BUILDING* on page 95.)

–B *bldhost-file*

Uses *bldhost-file* as the build hosts file for a parallel build. If you do not specify **–B**, **clearmake** uses the file **.bldhost.$CCASE_HOST_TYPE** in your home directory. When you use **–B**, you must also use **–J** or have the **CCASE_CONC** environment variable set. For more information, see the **bldhost** reference page.

–C *compat-mode*

(Compatibility) Invokes one of **clearmake**'s compatibility modes. (Alternatively, you can use environment variable **CCASE_MAKE_COMPAT** in a BOS file or in the environment to specify a compatibility mode.)

On UNIX systems, *compat-mode* can be one of the following:

sgismake Emulates the **smake(1)** native to IRIX systems. To define built-in make rules, **clearmake** reads file **/usr/include/make/system.mk** instead of *ccase-home-dir***/etc/builtin.mk**.

sgipmake Emulates the **pmake(1)** native to IRIX systems. To define built-in make rules, **clearmake** reads file **/usr/include/make/system.mk** instead of *ccase-home-dir***/etc/builtin.mk**.

sun Emulates the standard **make(1)** native to SunOS systems. **clearmake** defines built-in make rules in the following ways:

- If you specify **–r**, **clearmake** reads *ccase-home-dir***/etc/sunvars.mk**.
- If you do not specify **–r**, **clearmake** reads *ccase-home-dir***/etc/sunvars.mk** and *ccase-home-dir***/etc/sunbuiltin.mk**. If the current directory contains a **default.mk** file, **clearmake** reads it; otherwise, **clearmake** reads **/usr/share/lib/make/make.rules** (Solaris) or **/usr/include/make/default.mk** (SunOS).

aix Emulates the standard **make(1)** native to IBM AIX systems.

gnu Emulates the Free Software Foundation's **Gnu make** program. To define built-in make rules, **clearmake** reads file *ccase-home-dir***/etc/gnubuiltin.mk** instead of *ccase-home-dir***/etc/builtin.mk**.

std Invokes the standard **clearmake** with no compatibility mode enabled. Use this option to nullify a setting of the environment variable **CCASE_MAKE_COMPAT**.

The **–C** option is UNIX-platform independent. However, some modes try to read system-specific files, and if the files do not exist, the command fails. For more information on the platform-specific methods for dealing with this failure, see *ClearCase and MultiSite Release Notes*.

On Windows systems, *compat-mode* can be one of the following:

gnu Emulates the Free Software Foundation's **Gnu make** program. To define built-in make rules, **clearmake** reads file *ccase-home-dir***\etc\gnubuiltin.mk** instead of *ccase-home-dir***\etc\builtin.mk**.

std Invokes the standard **clearmake** with no compatibility mode enabled. Use this option to nullify a setting of the environment variable **CCASE_MAKE_COMPAT**.

For details on compatibility mode features, see *Using clearmake Compatibility Modes* in *Building Software with ClearCase*.

–V

(View) Restricts configuration lookup to the current view only. Winkin is disabled. This option is mutually exclusive with **–M**.

–M

(Makefile) Restricts dependency checking to makefile dependencies only—those dependencies declared explicitly in the makefile or inferred from a suffix rule. All detected dependencies are ignored. For safety, this disables winkin.

For example, a derived object in your view may be reused even if it was built with a different version of a header file than is currently selected by your view. This option is mutually exclusive with **–V**.

–O (Objects)
–T (Time stamps)
–F (Files)

(**–O**, **–T**, and **–F** are mutually exclusive.)

–O compares only the names and versions of objects listed in the targets' CRs; it does not compare build scripts or build options. This is useful when this extra level of checking would force a rebuild that you do not want. Examples:

- The only change from the previous build is the setting or canceling of a "compile-for-debugging" option.
- A target was built using a makefile in the current working directory. Now, you want to reuse it in a build to be performed in the parent directory, where a different makefile builds the target (with a different script, which typically references the target using a different pathname).

–T makes rebuild decisions using the standard algorithm, based on time stamps; configuration lookup is disabled. (A CR is still created for each build script execution.)

NOTE: This option causes both view-private files and derived objects to be used for build avoidance. Because the view-private file does not have a CR to be included in the CR hierarchy, the hierarchy created for a hierarchical build has a gap wherever **clearmake** reuses a view-private file for a subtarget.

–F works like **–T**, but also suppresses creation of configuration records. All MVFS files created during the build are view-private files, not derived objects.

–A *BOS-file ...*
You can use this option one or more times to specify BOS files to be read instead of, or immediately after, the ones that are read by default. Using **–N** along with this option specifies "instead of"; omitting **–N** causes **clearmake** to read the **–A** file after reading the standard BOS files.

Alternatively, you can specify a colon-separated list of BOS file pathnames (UNIX) or a semicolon-separated list such pathnames as the value of environment variable **CCASE_OPTS_SPECS**.

–ver·sion
Prints version information about the **clearmake** executable.

–VerAll
Prints version information about the **clearmake** executable and the libraries (UNIX) or ClearCase DLLs (Windows) that **clearmake** uses.

MAKE MACROS AND ENVIRONMENT VARIABLES

String-valued variables called make macros can be used anywhere in a makefile: in target lists, in dependency lists, and/or in build scripts. For example, the value of make macro CFLAGS can be incorporated into a build script as follows:

- UNIX:

```
cc -c $(CFLAGS) msg.c
```

- Windows:

```
cl $(CFLAGS) msg.c
```

Environment variables (EVs) can also be used in a makefile, but only in a build script. For example:

```
print:
     print_report -style $$PRT_STYLE -dest $${PRT_DEST}.rpt
```

clearmake converts the double-dollar-sign (**$$**) to a single dollar sign:

- On UNIX systems, the EV is expanded in the shell in which the build script executes. (Programs invoked by the build script can also read their environment, using **getenv(2)**.)
- On Windows systems, the EV is expanded in the shell in which the build script executes only if the shell recognizes the dollar sign ($) (**cmd.exe** does not).

Conflict Resolution

Conflicts can occur in specifications of make macros and environment variables. For example, the same make macro might be specified both in a makefile and on the command line; or the same name may be specified both as a make macro and as an environment variable.

clearmake resolves such conflicts similarly to other **make** variants; it uses the following priority order, from highest to lowest:

1. Target-specific macros specified in a BOS file
2. Target-specific macros specified in a makefile
3. Make macros specified on the command line
4. Make macros specified in a BOS file
5. Make macros specified in a makefile
6. Environment variables
7. Built-in macros

Using the **–e** option gives environment variables higher priority than make macros specified in a makefile.

Conflict Resolution Details. The following discussion treats this topic more precisely but less concisely.

clearmake starts by converting all EVs in its environment to make macros. (**SHELL** is an exception—see *SHELL Environment Variable* on page 104.) These EVs are also placed in the environment of the shell process in which a build script executes. Then, it adds in the make macros declared in the makefile. If this produces name conflicts, they are resolved as follows:

- If **clearmake** was not invoked with the **–e** option, the macro value overwrites the EV value in the environment.
- If **clearmake** was invoked with the **–e** option, the EV value becomes the value of the make macro.

Finally, **clearmake** adds make macros specified on the command line or in a BOS file; these settings are also added to the environment. These assignments *always* override any others that conflict. (A command-line assignment overrides a BOS setting of the same macro.)

SHELL Environment Variable

clearmake does not use the **SHELL** environment variable to select the shell program in which to execute build scripts. It uses a UNIX Bourne shell (**/bin/sh**) or Windows **cmd.exe**, unless you specify another program with a **SHELL** macro. You can specify **SHELL** on the command line, in the makefile, or in a build options spec; the value of **SHELL** must be a full pathname, and on Windows, it must include the file extension.

NOTE: If **clearmake** determines that it can execute the build script directly, it does not use the shell program even if you specify one explicitly. If you use Windows **.bat** files in build scripts, you must make them executable (use the **cleartool protect** command). To force **clearmake** to always use the shell program, set the environment variable **CCASE_SHELL_REQUIRED**.

Specifying Command Options in an Environment Variable

The **CCASE_MAKEFLAGS** and **MAKEFLAGS** environment variables provide an alternative (or supplementary) mechanism for specifying **clearmake** command options. These environment variables can contain a string of keyletters, the same letters used for **clearmake** command-line options. (However, **clearmake** does not allow options that take arguments in a **CCASE_MAKEFLAGS** or **MAKEFLAGS** string. See *Special Environment Variables* for information about specifying options that are not supported through **CCASE_MAKEFLAGS** or **MAKEFLAGS**.)

For example, the commands

setenv CCASE_MAKEFLAGS ei *(options set in environment)*
clearmake foo

are equivalent to this one:

clearmake -ei foo *(options set on command line)*

clearmake combines the value of **CCASE_MAKEFLAGS** or **MAKEFLAGS** with the options specified on the command line (if any). The combined string of keyletters becomes the value of the macro **MAKEFLAGS**, available to build scripts.

This is very useful for build scripts that involve recursive invocations of **clearmake**. When **clearmake –n** is applied to such a build script, all the nested invocations of **clearmake** pick up the "no-execute" option from the value of **CCASE_MAKEFLAGS** or **MAKEFLAGS**. Thus, no targets

are actually rebuilt, even though many levels of **clearmake** command may be executed. This is one way to debug all of the makefiles for a software project without building anything.

clearmake uses one of **CCASE_MAKEFLAGS** or **MAKEFLAGS**, but not both. If **CCASE_MAKEFLAGS** is set, **clearmake** uses it. If **CCASE_MAKEFLAGS** is not set, **clearmake** looks for **MAKEFLAGS**.

If you use other **make** programs in addition to **clearmake**, putting **clearmake**-specific options in the **MAKEFLAGS** environment variable may cause the **make** programs to generate errors. Therefore, we suggest you use the **CCASE_MAKEFLAGS** and **MAKEFLAGS** environment variables in the following ways:

If you use...	**Use...**
clearmake only	**CCASE_MAKEFLAGS**
clearmake and other **make** programs, but do not use **clearmake**-specific options	**MAKEFLAGS**
clearmake and other **make** programs, and do use **clearmake**-specific options	**CCASE_MAKEFLAGS** (all options) for **clearmake** builds **MAKEFLAGS** (all options except **clearmake**-specific options) for other **make** builds

Special Environment Variables

The environment variables described below are also read by **clearmake** at startup. In some cases, as noted, you can also specify the information as a make macro on the command line, in a makefile, or in a BOS file.

CCASE_ABE_PN (or **CLEARCASE_ABE_PN**)
: The full pathname with which **clearmake** invokes the audited build executor (**abe**) on a local or remote host during a parallel build.

  Default: **/bin/abe**

CCASE_AUDIT_TMPDIR (or **CLEARCASE_BLD_AUDIT_TMPDIR**)
: Sets the directory where **clearmake** and **clearaudit** create temporary build audit files. If this variable is not set or is set to an empty value, **clearmake** creates these files in the directory specified by the **TMPDIR** (UNIX) or **TMP** (Windows) environment variable.

  On UNIX systems, if neither **CCASE_AUDIT_TMPDIR** nor **CLEARCASE_BLD_AUDIT_TMPDIR** is set, **clearmake** creates these files in the **/tmp** directory.

  All temporary files are deleted when **clearmake** exits. If the value of **CCASE_AUDIT_TMPDIR** is a directory under a VOB-tag, **clearmake** prints an error message and exits.

NOTE: Multiple build clients can use a common directory for audit files. Names of audit files are unique because **clearmake** names them using both the PID of the **clearmake** process and the hostname of the machine on which the process is running.

Default on UNIX: **/tmp**

Default on Windows: None

CCASE_AUTO_DO_CI

Checks in DOs checked out by **clearmake –c** unless the build of the corresponding target fails or the automatic checkout of the DO or a sibling DO fails. Checkout comments are preserved. The **checkin** is invoked with the **–ptime** option to preserve the DO's modification time. This environment variable has no effect unless you specify **–c**.

Default: Undefined

CCASE_BLD_HOSTS

Specifies one or more build hosts on which **clearmake** must build targets. For more information, see *UNIX Only—Building Targets on Specified Hosts* on page 96.

Default: Undefined.

CCASE_BLD_NOWAIT

Turns off **clearmake**'s sleep-check cycle during a build. When this environment variable is set, **clearmake** does not check for a VOB-lock (or wait for the VOB to be unlocked). See *clearmake SLEEP* on page 93 for more information.

CCASE_BLD_UMASK (or **CLEARCASE_BLD_UMASK**)

Sets the **umask(1)** value to be used for files created from a **clearmake** build script. It may be advisable to have this EV be more permissive than your standard umask—for example, **CCASE_BLD_UMASK** = 2 where umask = 2. The reason to create DOs that are more accessible than other files is winkin: a winked-in file retains its original ownership and permissions. For example, when another user winks in a file that you originally built, the file is still owned by you, is still a member of your principal group, and still has the permissions with which you created it. You can use the standard **chmod** command to change the permissions of a DO after you create it, and these permissions remain in effect while the DO is unshared. However, for a shared DO, you may need to use the standard **chmod** and **protect –chmod** to set appropriate permissions.

If you are using a tool that ignores **umask** (and hence **CCASE_BLD_UMASK**) settings and you want winkins to work correctly, you have to use **chmod** on the file in your build script to give it write permissions if the tool creates the file without these permissions.

CCASE_BLD_UMASK can also be coded as a make macro.

NOTE: If you want to use **CCASE_BLD_UMASK**, do not set your **umask** value in your shell startup file. If you set the **umask** value in your startup file, the **umask** value will be reset

to its original value when clearmake starts a shell to run the build script. Setting **CCASE_BLD_UMASK** in your startup file has no effect.

Default: Same as current umask.

CCASE_BLD_VOBS
A list of VOB-tags (separated with a space, tab, comma, colon (UNIX), or semicolon (Windows) to be checked for lock status during a build. If a VOB on this list is locked, **clearmake** goes into a sleep-check cycle. See *clearmake SLEEP* on page 93 for more information.

CCASE_CONC (or **CLEARCASE_BLD_CONC**)
Sets the concurrency level. This EV takes the same values as the **–J** option. Specifying a **–J** option on the command line overrides the setting of this EV.

Default: None.

CCASE_DNVW_RETRY
Specifies time-out period, in minutes, for **clearmake** to wait before trying to contact an inaccessible view listed in its cache. To disable the cache, set **CCASE_DNVW_RETRY** to 0.

Default: 60 minutes.

CCASE_HOST_TYPE (or **CLEARCASE_BLD_HOST_TYPE**)
Determines the name of the build hosts file to be used during a parallel build (**–J** option): file **.bldhost.$CCASE_HOST_TYPE** in your home directory. (Your home directory is determined by examining the password database.)

Specifying a **–B** option on the command line overrides the setting of this EV.

C Shell Users: Set this EV in your **.cshrc** file, not in your **.login** file. The parallel build facility invokes a remote shell, which does not read the **.login** file.

CCASE_HOST_TYPE can also be coded as a make macro.

Default: none.

CCASE_MAKE_CFG_DIR (or **CLEARCASE_MAKE_CONFIG_DIR**)
Expands to the full pathname of the **clearmake** configuration directory in the ClearCase installation area—typically **/usr/atria/config/clearmake** (UNIX) or *ccase-home-dir***config****clearmake** (Windows).

CCASE_MAKE_COMPAT (or **CLEARCASE_MAKE_COMPAT**)
Specifies one of **clearmake**'s compatibility modes. This EV takes the same values as the **–C** option. Specifying a **–C** option on the command line overrides the setting of this EV. This EV may also be coded as a make macro, but only in a BOS file (not in a makefile).

Default: None.

CCASE_OPTS_SPECS (or **CLEARCASE_BLD_OPTIONS_SPECS**)
A list of pathnames separated by colons (UNIX) or semicolons (Windows), each of which specifies a BOS file to be read. You can use this EV instead of specifying BOS files on the command line with one or more **–A** options.

Default: None.

CCASE_SHELL_FLAGS (or **CLEARCASE_BLD_SHELL_FLAGS**)
Specifies command options to be passed to the subshell program that executes a build script command.

Default for UNIX: **–e**

Default for Windows: None

CCASE_SHELL_REQUIRED
Forces **clearmake** to execute build scripts in the shell program you specify with the **SHELL** macro. To make **clearmake** execute builds scripts in the shell program, set this EV to **TRUE**. To allow **clearmake** to execute build scripts directly, unset the EV.

Default: **clearmake** executes build scripts directly.

CCASE_VERBOSITY (or **CLEARCASE_BLD_VERBOSITY**)
Sets the **clearmake** message logging level, as follows:

1	Equivalent to **–v** on the command line
2	Equivalent to **–d** on the command line
0 or **undefined**	Equivalent to standard message logging level

If you also specify **–v** or **–d** on the command line, the higher value prevails.

Default: 0

CMAKE_PNAME_SEP
Sets the pathname separator for pathnames constructed by **clearmake**. This variable can also be coded as a make macro in the makefile or in a BOS file.

Default: If this variable is not set or is set to any other value than **/** or **** (the slashes), **clearmake** uses **** (a backslash) as the pathname separator.

EXAMPLES

- Unconditionally build the default target in a particular makefile, along with all its dependent targets.

  % **clearmake -u -f project.mk**

- Build target **hello** without checking build scripts or build options during configuration lookup. Be moderately verbose in generating status messages.

```
z:\src> clearmake -v -O hello
```

- Build the default target in the default makefile, with a particular value of make macro INCL_DIR. Base rebuild decisions on time-stamped comparisons instead of performing configuration lookup, but still produce CRs.

  ```
  y:\> clearmake -T INCL_DIR=\src\include_test
  ```

- Perform a parallel build of target **bgrs**, using up to five of the hosts listed in file **.bldhost.solaris** in your home directory.

  ```
  % setenv CCASE_HOST_TYPE solaris
  % clearmake -J 5 bgrs
  ```

- Build target **bgrs.exe**, restricting configuration lookup to the current view only. Have environment variables override makefile macro assignments.

  ```
  z:\src> clearmake -e -V bgrs.exe
  ```

- Build the default target in Sun compatibility mode.

  ```
  % clearmake -C sun
  ```

FILES

*ccase-home-dir***/etc/builtin.mk** (UNIX) or *ccase-home-dir***\etc\builtin.mk** (Windows)

SEE ALSO

abe, bldhost, bldserver.control, clearaudit, omake, promote_server, scrubber, umask(1)

Building Software with ClearCase

clearprojexp

Starts the UCM Project Explorer

APPLICABILITY

Product	Command Type
ClearCase	command
ClearCase LT	command

Platform
UNIX
Windows

SYNOPSIS

clearprojexp

DESCRIPTION

The **clearprojexp** command starts the Project Explorer, a graphical utility that lets you create, manage, work in or view UCM projects.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

None.

EXAMPLE

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

- Start the Project Explorer.

  clearprojexp

SEE ALSO

chbl, **chfolder**, **chproject**, **chstream**, **clearjoinproj**, **clearmrgman**, **deliver**, **mkbl**, **mkcomp**, **mkfolder**, **mkproject**, **mkstream**, **mkview**, **rebase**

deliver

Delivers changes in a UCM development stream to the project's integration stream

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

- Deliver changes in the development stream using the graphical user interface:

  deliver –g·raphical [**–str·eam** *stream-selector*] [**–to** *integration-view-tag*]

- Cancel or obtain the status of a deliver operation in progress:

  deliver { **–can·cel** | **–sta·tus** [**–l·ong**] } [**–str·eam** *stream-selector*]

- Preview a deliver operation:

  deliver –preview [**–s·hort** | **–l·ong**] [**–str·eam** *stream-selector*] [**–to** *integration-view-tag*]
  [**–act·ivities** *activity-selector* ...]

- Deliver changes in the development stream:

  deliver [**–str·eam** *stream-selector*] [**–to** *integration-view-tag*] [**–act·ivities** *activity-selector*[,...]]
  [**–com·plete**] [**–gm·erge** | **–ok**] [**–q·uery** | **–abo·rt** | **–qal·l**] [**–ser·ial**] [**–f·orce**]

- Resume or complete work on a deliver operation:

  deliver { **–res·ume** |**–com·plete** } [**–str·eam** *stream-selector*] [**–gm·erge** | **–ok**]
  [**–q·uery** | **–ab·ort** | **–qal·l**] [**–ser·ial**] [**–f·orce**]

DESCRIPTION

The **deliver** command lets you deliver work from your development stream to the project's integration stream. Work is delivered from your development stream to an integration view. There may be several steps to delivering work:

- Previewing the changes to be delivered
- Identifying the activities you want to deliver
- Resolving merge conflicts
- Testing and building work in the integration stream
- Completing a deliver operation, which checks in new versions and records other information.

If a deliver operation is interrupted through a system interrupt or user action, you must explicitly resume or cancel the deliver operation.

In general, it is good practice to check in all work to your development stream before beginning a deliver operation.

The Integration Activity

The deliver operation creates a UCM activity called the integration activity, which records a change set for the deliver operation. The activity name is of the form *deliver.stream-name.date-stamp*. When the deliver operation begins, the integration activity becomes the current activity for the integration view in use.

One-Step Deliver Operation

You can deliver your work in one step by specifying the **–complete** and **–force** options. The **–force** option suppresses prompting for user input during the deliver operation. The **–complete** option causes the deliver operation to continue to completion after the merge phase. Use this feature carefully to avoid the possibility of delivering merged files that may not compile.

Using deliver with MultiSite

The **deliver** command determines whether the integration stream and development stream are mastered at different replicas. If they are, a remote deliver operation is put into effect. The development stream is assigned a **posted** status.

After the stream is in the posted state, the deliver operation can be continued only by someone working at the integration stream's replica. Generally, this is the team's project integrator. Also, once posted, the deliver operation can be canceled only by a user at the replica where the integration stream resides.

The **deliver –status** command reports on any remote deliver operation in progress for the specified stream. Using this information, the project integrator can then cancel or continue the deliver operation, using the **–cancel** option to halt the deliver operation, or the **–resume** or **–complete** options to continue the deliver operation.

You can create activities and perform checkins and checkouts for your development stream while the remote deliver is in process. However, you cannot perform any of the following operations while a remote deliver operation is in progress:

- Add, remove, or create baselines
- Add or remove components
- Rebase the development stream
- Post another deliver operation.

RESTRICTIONS

Identities: No special identity required.

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

Mastership: The current replica must master the development streams.

OPTIONS AND ARGUMENTS

INVOKING THE GRAPHICAL USER INTERFACE: *Default*. Nongraphical interface.

–g·raphical
Invokes the graphical user interface for **deliver**.

SPECIFYING THE SOURCE AND DESTINATION FOR THE DELIVER OPERATION. *Default:* The source is the stream attached to the current view. The default destination is the integration stream of the development stream's project, using either a view attached to the integration stream owned by the current user, or the integration view used by the last deliver operation executed by the current user.

–stre·am *from-stream-selector*
Specifies a development stream that is the source for the deliver operation.

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

–to *integration-view-tag*
Specifies a view attached to the integration stream for the development stream's project.

CANCELLING A DELIVER OPERATION. *Default:* None.

–can·cel
Halts a deliver operation in progress, returning the source and destination streams to their states before the deliver operation began. However, this option cannot undo a deliver operation after the completion phase has begun.

Also use **–cancel** when a deliver operation is interrupted with CTRL+C or when it encounters an external error or condition that requires more information.

OBTAINING THE STATUS OF A DELIVER OPERATION. *Default:* None.

–sta·tus

Displays the status of a deliver operation. You are informed whether a deliver operation is in progress for the specified stream, whether the deliver is to a local stream or a remote stream, and, in the case of a remote deliver, whether the posted deliver has been merged with the integration stream.

PREVIEWING THE RESULTS OF A DELIVER OPERATION. *Default:* For each activity that would be delivered, displays the owner, activity-selector, and title.

–pre·view

Shows activities that would be delivered if you were to execute the deliver operation for the specified stream. These are any activities that have changed since the last deliver operation from this stream. Use –**preview** only when there is no deliver operation in progress for the stream.

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** or –**preview,** displays a list of versions that may require merging, in addition to the default information displayed by **–status** or –**preview**.

–**s·hort**

Modifies the –**preview** option. (Currently, this option does not modify the default **–preview** output.)

SELECTING ACTIVITIES TO DELIVER. *Default:* Delivers all activities in the stream that have changed since the last deliver operation from the stream.

–act·ivities *activity-selector, ...*

Specifies a list of activities to deliver. The list of activities must be self-consistent: they must not depend on the inclusion of any unspecified activities. For example, activity A2 is dependent on activity A1 if they both contain versions of the same element and A2 contains a later version than A1. Additionally, any activities that have been included in baselines but not delivered must also be delivered if there are changes for that component in the specified activities. If the list of activities you specify is incomplete, the additional required activities are listed and the operation fails.

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

RESUMING A DELIVER OPERATION. *Default:* None.

–res·ume

Resumes a deliver operation from the point at which it has been suspended.

COMPLETING A DELIVER OPERATION. *Default:* None.

–com·plete

Completes a deliver operation. Verifies that changes from the activities being delivered have been merged with versions in the project integration stream and that merge conflicts have been resolved. Checks in resulting new versions to the integration stream and records that the deliver operation has been made. If merge conflicts exist, the deliver operation is suspended.

Use this option to bring a deliver operation through the completion phase or to resume a suspended deliver operation. To complete a deliver operation, you must specify this option—checking in merged versions to the integration view alone does not complete the deliver operation.

When used for a deliver operation in progress, this option implies the –**resume** option—that is, **deliver –complete** reports any merges that are still required and attempts to resolve them.

MERGING. *Default:* Merging 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.

–gm·erge

Performs a graphical merge for each element that requires it. This option does not remain in effect after a deliver operation is interrupted.

–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 deliver operation is interrupted.

–q·uery

Turns off automated merging for nontrivial merges and prompts you for confirmation before proceeding with each 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 deliver operation is interrupted.

–abo·rt

Cancels a merge if it is not completely automatic. This option does not remain in effect after a deliver operation is interrupted.

–qal·l

Turns off all automated merging. Prompts you for confirmation before proceeding with each change. This option does not remain in effect after a deliver operation is interrupted.

–ser·ial

Use a serial format when reporting differences among files. Differences are presented in

a line-by-line comparison with each line from one contributor, instead of in a side-by-side format. This option does not remain in effect after a deliver operation is interrupted.

CONFIRMATION STEP. *Default:* Prompts for use input.

–f·orce

Suppresses prompting for user input during the course of a deliver operation. The **–force** option does not remain in effect if the deliver operation is interrupted. You must include it again on the command line when you restart the deliver operation with **–resume** or **–complete**. The merge options to the deliver command are not affected by the **–force** option.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Start a deliver operation using command defaults.

cmd-context **deliver -to webo_integ**

```
Changes to be DELIVERED:
   FROM: stream "chris_webo_dev"
   TO: stream "integration"
Using integration view: "webo_integ".
Do you wish to continue with this deliver operation?  [no] yes
Needs Merge "/view/webo_integ/vobs/webo_modeler/design/foo" [(automatic)
to /main/integration/1 from /main/integration/chris_webo_dev/1 (base also
/main/integration/1)]
Checked out "/view/webo_integ/vobs/webo_modeler/design/foo" from version
"/main/integration/1".
```

```
Attached activities:
activity:deliver.chris_webo_dev.20000606.160519@/vobs/webo_pvob  "deliver
chris_webo_dev on 06/06/00 16:05:19."

Needs Merge "/view/webo_integ/vobs/webo_modeler/design/foo" [to
/main/integration/CHECKEDOUT from /main/integration/chris_webo_dev/1 base
/main/integration/1]

Trivial merge: "/view/webo_integ/vobs/webo_modeler/design/foo" is same as
base "/view/webo_integ/vobs/webo_modeler/design/foo@@/main/integration/1".

Copying
"/view/webo_integ/vobs/webo_modeler/design/foo@@/main/integration/chris_we
bo_dev/1" to output file.
Moved contributor "/view/webo_integ/vobs/webo_modeler/design/foo" to
"/view/webo_integ/vobs/webo_modeler/design/foo.contrib".
Output of merge is in "/view/webo_integ/vobs/webo_modeler/design/foo".
Recorded merge of "/view/webo_integ/vobs/webo_modeler/design/foo".

Deliver has merged
FROM: stream "chris_webo_dev"
TO: stream "integration"
Using integration view: "webo_integ".
Build and test are necessary in integration view "webo_integ"
to ensure that the merges were completed correctly.  When build and
test are confirmed, run "cleartool deliver -complete".
```

- Complete a deliver operation that is in progress.

  cmd-context **deliver –complete**

```
Resume deliver
   FROM: stream "chris_webo_dev"
   TO: stream "integration"
Using integration view: "webo_integ".
Do you wish to continue with this deliver operation?  [no] yes
Are you sure you want to complete this deliver operation?  [no] yes
Deliver has completed
   FROM: stream "chris_webo_dev"
   TO: stream "integration"
Using integration view: "webo_integ".
```

- Check the status of a deliver operation.

  cmd-context **deliver –status**

```
Deliver operation in progress on stream "stream:chris_webo_dev@\webo_pvob"
   Started by "ktessier" on "14-Jun-00.16:07:46"
   Using integration activity "deliver.chris_webo_dev.20000614.160746".
   Using view "webo_integ".
   Activities will be delivered to stream "stream:integration@\webo_pvob".
Development Stream Baselines:
baseline:deliverbl.chris_webo_dev.20000614.160746.129@\webo_pvob
Activities:
activity:fix_copyright@\webo_pvob
activity:update_date@\webo_pvob
activity:fix_defect_215@\webo_pvob
```

- Cancel a deliver operation that is in progress.

  cmd-context **deliver –cancel**

```
Cancel deliver
   FROM: stream "chris_webo_dev"
   TO: stream "integration"
Using integration view: "webo_integ".
Are you sure you want to cancel this deliver operation?  [no] yes
Private version of "/view/webo_integ/vobs/webo_modeler/design/add_proc"
saved in "/view/webo_integ/vobs/webo_modeler/design/add_proc.keep".
Deliver of stream "chris_webo_dev" canceled.
```

SEE ALSO

checkin, **checkout**, **findmerge**, **rebase**, **setactivity**

diffbl

Compares the contents of UCM baselines or streams

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

- Display differences between two baselines or streams nongraphically:

  diffbl [–act·ivities] [–ver·sions] [–fir·st_only]
  { *baseline-selector1* | *stream-selector1* }
  { *baseline-selector2* | *stream-selector2* }

- Display differences between the specified baseline and its predecessor baseline nongraphically:

  diffbl –pre·decessor [–act·ivities] [–ver·sions] *baseline-selector*

- Display differences between two baselines graphically:

  diffbl –g·raphical *baseline-selector1 baseline-selector2*

- Display differences between the specified baseline and its predecessor baseline graphically:

  diffbl –g·raphical –pre·decessor *baseline-selector*

DESCRIPTION

The **diffbl** command compares the contents of two baselines or streams and displays any differences it finds. You can choose to see differences in terms of activities or versions, or both.

You can use the **diffbl** command to compare a baseline and a stream, a baseline and a baseline, or a stream and a stream. When specifying a stream, all baselines in the stream are used in the comparison as well as any changes in the stream that are not yet captured in a baseline.

The **diffbl** command must be issued from a view context to display versions. The view context is needed to resolve pathnames of versions.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SPECIFYING THE INFORMATION TO DISPLAY. *Default:* **–activities**.

–act·ivities
Displays differences in terms of activities.

–ver·sions
Displays differences in terms of versions.

–fir·st_only
Shows only those changes that appear in the first object specified for the comparison.

COMPARING A VERSION WITH ITS PREDECESSOR. *Default:* None.

–pre·decessor
Displays differences between the specified baseline and its immediate predecessor.

REPORTING DIFFERENCES GRAPHICALLY. *Default:* Reports differences in nongraphical form.

–g·raphical
Displays differences graphically. This only applies to baselines and not streams.

SELECTING THE OBJECTS TO COMPARE. *Default:* None.

baseline-selector1
stream-selector1
Specifies an object to use in the comparison.

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

baseline-selector2
stream-selector2
Specifies an object to use in the comparison.

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

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Compare activities in two streams:

cmd-context **diffbl stream:java_int stream:java_dev**

```
<< deliver.java_dev.19990917.140443 "deliver java_dev on 09/17/99
14:04:43."

<< deliver.java_dev.19990917.141046 "deliver java_dev on 09/17/99
14:10:46."
```

- Compare baselines in two streams:

  cmd-context **diffbl -ver stream:java_int stream:java_dev**

  ```
  << /vobs/parser/myfile.c@@/main/java_int/2

  << /vobs/parser/myfile.c@@/main/java_int/3
  ```

- Compare a baselines with its predecessor baseline:

  cmd-context **diffbl -predecessor test_sum.D001207.124@net/acrolein/export/home/lli/vobs/lli_test_sum**

  ```
  >> deliver.lli_test_sum.20001113.165650 “deliver lli_test6_sum on 11/13/00
  16:56:50.”

  >> test_act_00795 “test act”
  ```

SEE ALSO

chbl, **lsbl**, **mkbl**, **rmbl**

lsactivity

Lists information about UCM activities

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsact·ivity [**–s·hort** | **–l·ong** | **–fmt** *format-string* |
–anc·estor [**–fmt** *format-string*] [**–dep·th** *depth*]]
[**–inv·ob** *vob-selector* | **–in** *stream-selector-name* |
–cac·t | [**–cac·t**] **–vie·w** *view-tag* | **–cvi·ew** | *activity-selector* ...] [**–obs·olete**]

DESCRIPTION

The **lsactivity** command lists information about UCM activities.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SPECIFYING OUTPUT FORMAT *Default*: A one-line summary of the activity.

–s·hort
Displays only the name of each activity.

–l·ong
Displays a detailed description of each activity.

–fmt *format-string*
Displays information in the format specified by *format-string*. See the **fmt_ccase** reference page.

–anc·estor [–**fmt** *format-string* **] [** –**dep·th** *depth* **]**
Displays the containing stream, project, and folder for one or more activities. For information on the **–fmt** option, see the **fmt_ccase** reference page. The **–depth** option sets the number of levels displayed. The *depth* argument must be a positive integer.

SPECIFYING THE ACTIVITY. *Default:* **–cview**.

–inv·ob *vob-selector*
Displays a list of all activities in the specified project VOB.

–in *stream-selector*
Displays a list of all activities in the specified stream.

–cac·t
Displays information for the current activity.

–vie·w *view-tag*
For the specified view, displays a list of all activities in its stream.

–cvi·ew
For the current view, displays a list of all activities in its stream.

activity-selector ...
Specifies one or more activities to list.

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.

LISTING OBSOLETE ACTIVITIES. *Default*: List only nonobsolete activities.

–obs·olete
Includes obsolete activities in the listing. Obsolete activities are those that have been processed with **lock –obsolete**.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive

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

NOTE: In the UNIX 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.

Display detailed information for an activity.

cmd-context **lsactivity -l fix_copyright**

```
activity "fix_copyright"
06-Jun-00.15:49:23 by Ken Tessier (ktessier.user@mymachine)
owner: ktessier
group: user
stream: chris_webo_dev@/vobs/webo_pvob
title: Fix copyright text
change set versions:
/vobs/webo_modeler/design/add_proc@@/main/chris_webo_dev/1
/vobs/webo_modeler/design/foo@@/main/integration/chris_webo_dev/1
```

- Display a short description of the current activity. This is the currently set activity for the view from which the command was issued.

  cmd-context **lsact -cact**

  ```
  06-Jun-00.17:16:12  update_date  ktessier   "Update for new date
  convention"
  ```

SEE ALSO

chactivity, **lock**, **mkactivity**, **rmactivity**

lsactivity

lsbl

Lists information about a UCM baseline

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

- List baseline information per stream or component or by promotion level:
  lsbl [–s·hort | –l·ong | –fmt *format-string* **| –tre·e]**
  [–lev·el *promotion-level* **| [–ltl·evel** *promotion-level* **] [–gtl·evel** *promotion-level* **]]**
  [–str·eam *stream-selector* **| –com·ponent** *component-selector* **] [–obs·olete]**
- List information for one or more specific baselines:
  lsbl [–s·hort | –l·ong | –fmt *format-string* **] [–tre·e] [** *baseline-selector ...* **] [–obs·olete]**

DESCRIPTION

The **lsbl** command lists information for one or more UCM baselines.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SPECIFYING THE OUTPUT. *Default:* A one-line summary of each baseline.

–s·hort
Displays only the name of each baseline.

–l·ong
Displays detailed information for each baseline, including ownership, creation, and label information and the UCM stream, component, change sets, and promotion level associated with the baseline.

–fmt *format-string*
Displays information in the specified format. See the **fmt_ccase** reference page for details.

–tre·e
Displays a list of streams and baselines associated with one or more baselines. The list is indented to show the order of succession for baselines.

FILTERING BY PROMOTION LEVEL. *Default:* All promotion levels.

–lev·el *promotion-level*
Displays a list of baselines that are at the specified promotion level. An error results if the specified level is not in the project VOB's current list of valid promotion levels. This option modifies the **–stream** and **–component** options. For general information on promotion levels, see the **setplevel** reference page.

–ltl·evel *promotion-level*
Displays a list of baselines whose promotion level is lower than the one specified by the promotion-level argument. For example, if your project has four promotion levels in this order: **PROTOTYPE**, **REVIEWED**, **TESTED**, **CERTIFIED**, and you use the argument **–ltlevel TESTED**, the **lsbl** command displays a list of all baselines whose promotion level is **PROTOTYPE** or **REVIEWED**. This option modifies the **–stream** and **–component** options.

–gtl·evel *promotion-level*
Displays a list of baselines whose promotion level is greater than the one given. This option modifies the **–stream** and **–component** options.

SPECIFYING THE BASELINE. *Default:* Baselines in the UCM project VOB of the current directory.

–str·eam *stream-selector*
Displays a list of baselines created in the specified stream.

–com·ponent *component-selector*
Displays a list of baselines of the specified component.

baseline-selector ...
Specifies one or more baselines for which information is displayed.

LISTING OBSOLETE BASELINES. *Default*: List only nonobsolete baselines.

–obs·olete
Includes obsolete baselines in the listing. Obsolete baselines are those that have been processed with **lock –obsolete**.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Display a one-line summary (the default) of baselines of the specified component.

cmd-context **lsbl -component parser@/vobs/core_projects**

```
17-Sep-99.12:06:59  parser_INITIAL.112  bill   "parser_INITIAL"
  component: parser
```

- Display a description of baselines created in a stream:

  cmd-context **lsbl -stream java_int@/vobs/core_projects**

```
17-Sep-99.13:56:10  testbl.121  bill   "testbl"
  stream: java_int
  component: parser
17-Sep-99.14:05:30  new_bl.121  bill   "new_bl"
  stream: java_int
  component: parser
```

SEE ALSO

chbl, **deliver**, **describe**, **diffbl**, **lock**, **mkbl**, **rebase**, **rmbl**, **setplevel**

lsbl

lscomp

Lists information for a UCM component

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

lscomp [**–s**·**hort** | **–l**·**ong** | **–fmt** *format-string* | **–tre**·**e**]
[**–inv**·**ob** *vob-selector* | *component-selector ...*] [**–obs**·**olete**]

DESCRIPTION

The **lscomp** command lists information describing one or more UCM components.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SPECIFYING THE OUTPUT. *Default:* A one-line summary.

–s·**hort**
Displays only the name of each component.

–l·**ong**
Displays an expanded multiple-line listing for each component, similar to the **describe –long** command.

–fmt *format-string*
Displays information using the specified *format-string*. See the **fmt_ccase** reference page for details.

–tre·e
Recursively lists baselines and streams in the specified components. Output format is similar to that of the **lsvtree** command.

SPECIFYING THE COMPONENT *Default:* All components in the project VOB of the current directory.

–inv·ob *vob-selector*
Displays information for all components in the specified project VOB.

component-selector ...
Specifies one or more components for which information is displayed.

LISTING OBSOLETE COMPONENTS. *Default*: List only nonobsolete components.

–obs·olete
Includes obsolete components in the listing. Obsolete components are those that have been processed with **lock –obsolete**.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Display a description of components in the specified VOB.

cmd-context **lscomp -invob /vobs/projects**

```
17-Sep-99.12:06:59  parser  bill   "parser"
   root directory: "/vobs/parser"
29-Mar-99.17:23:16  applets  pklenk   "applets"
   root directory: "/vobs/applets"
29-Mar-99.17:23:25  booch  pklenk   "booch"
   root directory: "/vobs/booch"
29-Mar-99.17:23:37  libobj  pklenk   "libobj"
   root directory: "/vobs/libobj"
29-Mar-99.17:23:44  stage  pklenk   "stage"
   root directory: "/vobs/stage"
29-Mar-99.17:23:50  sun5_stage  pklenk   "sun5_stage"
   root directory: "/vobs/sun5_stage"
29-Mar-99.17:24:01  nt_i386_stage  pklenk   "nt_i386_stage"
   root directory: "/vobs/nt_i386_stage"
29-Mar-99.17:24:57  sys  pklenk   "sys"
   root directory: "/vobs/sys"
```

SEE ALSO

describe, **lsbl**, **mkcomp**, **rmcomp**

lscomp

lsfolder

Lists information about UCM folders

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsfolder [**–s·hort** | **–l·ong** | **–fmt** *format-string* |
–tre·e [**–fmt** *format-string*] [**–dep·th** *depth*] |
–anc·estor [**–fmt** *format-string*] [**–dep·th** *depth*]]
[**–inv·ob** *vob-selector* | **–in** *folder-selector* |
–vie·w *view-tag* | **–cvi·ew** | *folder-selector ...*] [**–obs·olete**]

DESCRIPTION

The **lsfolder** command displays information describing one or more UCM folders.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

CHOOSING A DISPLAY FORMAT. *Default*: A one-line summary.

–s·hort
Displays only the the name of each folder.

–l·ong
Displays a detailed listing for a folder.

–fmt *format-string*
Displays information in the specified format. See the **fmt_ccase** reference page for further information.

–tre·e [–**fmt** *format-string*] [–**dep·th** *depth*]
Displays information about a folder and its contents. Default output format is similar to that of the **lsvtree** command.

The **–fmt** option displays information in the format specified by the *format-string* argument. See the **fmt_ccase** reference page for details.

The **–depth** option lists the hierarchy of objects to the level specified by the *depth* argument. The *depth* argument must be a positive integer.

–**anc·estor** [–**fmt** *format-string*] [–**dep·th** *depth*]
Displays information about a folder and any parent folders.

The **–fmt** option formats information using the specified *format-string*. See the **fmt_ccase** reference page for further information.

The **–depth** option specifies how many levels to display. The *depth* argument must be a positive integer: a value of zero lists the entire hierarchy.

SPECIFYING A FOLDER. *Default:* All folders in the project VOB of the current directory.

–**inv·ob** *vob-selector*
Displays a list of folders in the specified project VOB.

–**in** *folder-selector ...*
Displays a list of subfolders of the specified folder or folders.

–vie·w *view-tag*
Displays information about the parent folder of the stream attached to the specified view.

–**cvi·ew**
Displays information about the parent folder of the stream attached to the current view.

folder-selector ...
Specifies one or more folders to list.

LISTING OBSOLETE FOLDERS. *Default*: List only nonobsolete folders.

–obs·olete
Includes obsolete folders in the listing. Obsolete folders are those that have been processed with **lock –obsolete**.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Display a one-line summary of the specified folder.

cmd-context **lsfolder Core_Parsers@/vobs/core_projects**

```
17-Sep-99.11:21:36  Core_Parsers  bill   "Core_Parsers"
```

- Display a long listing for the specfied folder.

  cmd-context **lsfolder -long RootFolder@/vobs/core_projects**

```
folder "RootFolder"
 17-Sep-99.10:52:34 by Bill Marrs (bill.user@propane)
  "Predefined Root folder."
  owner: bill
  group: user
  title: Root folder
  contains folders:
    Parsers
    Core_Parsers
  contains projects:
    Java_Parser
```

SEE ALSO

chfolder, **lock**, **mkfolder**, **rmfolder**

lsfolder

lsproject

Lists information about a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsproj·ect [–s·hort | **–l·ong** | **–fmt** *format-string* |
–tre·e [–fmt *format-string* **] [–dep·th** *depth* **]** |
–anc·estor [–fmt *format-string* **] [–dep·th** *depth* **]]**
[–inv·ob *vob-selector* | **–in** *folder-selector* |
–vie·w *view-tag* | **–cvi·ew** | *project-selector ...* **]**

DESCRIPTION

The **lsproject** command lists information for one or more UCM projects.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SELECTING A DISPLAY FORMAT. *Default*: A one-line summary.

–s·hort
Displays project names only.

–l·ong
Displays a detailed listing for a project, similar to the **describe –long** command.

–fmt *format-string*
Displays information in the specified format. See the **fmt_ccase** reference page for details.

–tre·e [–fmt *format-string*] [**–dep·th** *depth*]
Displays information for a project, including its hierarchy of streams and activities. By default output is presented in a version-tree format. You can modify how information is displayed with the **–fmt** and –**depth** options.

The **–fmt** option displays information using the specified format. See the **fmt_ccase** reference page for more information.

The **–depth** option lists the hierarchy of objects to the level specified by the *depth* argument. The *depth* argument must be a positive integer.

–anc·estor [–**fmt** *format-string*] [**–dep·th** *depth*]
Displays information about one or more projects and its parent folders.

The **–fmt** option displays information using the specified format. See the **fmt_ccase** reference page for further information.

The **–depth** option lists the hierarchy of objects to the level specified by the *depth* argument. The *depth* argument must be a positive integer.

SPECIFYING THE PROJECT. *Default:* All projects in the project VOB of the current directory.

–inv·ob *vob-selector*
Displays a list of all projects in the specified project VOB.

–in *folder-selector*
Displays a list of projects in the specified folder.

–**vie·w** *view-tag*
Displays information for the project containing the stream attached to the specified view.

–**cvi·ew**
Displays information for the project containing the stream attached to the current view.

project-selector ...
Specifies one or more projects to list.

LISTING OBSOLETE PROJECTS. *Default*: List only nonobsolete projects.

–obs·olete
Includes obsolete projects in the listing. Obsolete projects are those that have been processed with **lock –obsolete**.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Display a detailed description of the specified project.

cmd-context **lsproject -long Java_Parser_Project_28174@/vobs/core_projects**

```
project "Java_Parser_Project_28174"
17-Sep-99.11:24:18 by BillM (bill.user@propane)
  owner: bill
  group: user
  folder: RootFolder
  title: Java Parser Project
  development streams:
  modifiable components:
  default rebase promotion level: TESTED
  recommended baselines:
  policies:
    UnixIntVSnap disabled
    UnixDevVSnap disabled
    WinIntVSnap disabled
    WinDevVSnap disabled
    DeliverReqRebase disabled
    DeliverAllowNCoDevStr disabled
```

- Display a one-line summary of the project visible from the specified view.

  cmd-context **lsproject –view java_int**

  ```
  17-Sep-99.11:24:18 Java_Parser_Project_28174 bill "Java Parser Project"
  ```

SEE ALSO

chproject, **mkproject**, **rmproject**

lsproject

lsstream

Lists information about one or more UCM streams

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsstream [**–s**·**hort** | **–l**·**ong** | **–fmt** *format-string*
| –**tre**·**e** [**–fmt** *format-string*] [**–dep**·**th** *depth*]
| –**anc**·**estor** [**–fmt** *format-string*] [**–dep**·**th** *depth*]]
[**–inv**·**ob** *vob-selector* | –**in** *project-selector* | –**vie**·**w** *view-tag*
| –**cvi**·**ew** |*stream-selector* ...] [**–obs**·**olete**]

DESCRIPTION

The **lsstream** command displays information about one or more streams.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SELECTING A DISPLAY FORMAT. *Default*: One-line summary.

–s·**hort**
Displays only the name of each stream.

–l·**ong**
Displays detailed information for each stream, including the project it's associated with and the stream's name and title, activities, and foundation baselines.

–fmt *format-string*
Displays information in the specified format. See the **fmt_ccase** reference page for details.

–tre·e [–**fmt** *format-string*] [–**dep·th** *depth*]
Displays information for a stream, including its hierarchy of streams and activities. By default output is presented in a version-tree format. You can modify how information is displayed with the **–fmt** and –**depth** options.

The –**fmt** option presents information using the specified format string. See the **fmt_ccase** reference page for further information.

The **–depth** option specifies the number of levels displayed. The *depth* argument must be a positive integer.

–anc·estor [–**fmt** *format-string*] [–**dep·th** *depth*]
Displays information for a stream, including its containing project and folders. By default, output is presented in a version-tree format. You can modify how information is displayed with the **–fmt** and –**depth** options.

The –**fmt** option presents information using the specified format string. See the **fmt_ccase** reference page for further information.

The **–depth** option sets the number of levels displayed. The *depth* argument must be a positive integer.

SPECIFYING THE STREAM. *Default:* –**cview**.

–inv·ob *vob-selector*
Displays a list of all streams in the specified UCM project VOB.

–in *project-selector*
Displays a list of all streams for the specified project and highlights the integration stream.

–**vie·w** *view-tag*
Displays information for the stream connected to the specified view.

–**cvi·ew**
Displays information for the stream connected to the current view.

stream-selector ...
Displays information for specified stream or streams.

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.

LISTING OBSOLETE STREAMS. *Default*: List only nonobsolete streams.

–obs·olete
Includes obsolete streams in the listing. Obsolete streams are those that have been processed with **lock –obsolete**.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

Display a one-line summary of the stream attached to the specified view.

cmd-context **lsstream –view java_int**

```
17-Sep-99.11:54:50  java_int  bill   "Deliver your changes here"
```

- For all streams in the project VOB, display a detailed listing for the current stream using a tree format. The asterisk (*) indicates **java_int** is the stream attached to the current view.

  % **cd /vobs/core_projects**

  cmd-context **lsstream -tree**

```
*java_int                        stream     "Deliver your changes here"
   rebase.java_int.19990917.132524       activity   "rebase Deliver your
    changes here on 09/17/99 13:25:24."
   activity990917.133218        activity   "activity990917.133218"
   activity990917.133255        activity   "my new activity"
   new_activity                 activity   "new_activity"
   toms_edit                    activity   "toms_edit"
   activity990917.134751        activity   "activity990917.134751"
   deliver.java_dev.19990917.140443      activity   "deliver java_dev
    on 09/17/99 14:04:43."
   deliver.java_dev.19990917.141046      activity   "deliver java_dev
    on 09/17/99 14:10:46."
```

```
    java_dev                        stream     "java_dev"
       activity990917.140331         activity   "activity990917.140331"
```

SEE ALSO

chstream, **lock**, **mkstream**, **rmstream**

mkactivity

Creates a UCM activity

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

SYNOPSIS

mkact·ivity [–c·omment *c̲omment* | **–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*.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if there are locks on the following objects: the 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

mkactivity

mkbl

Creates a UCM baseline or set of baselines

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

Locks: An error is generated if there are locks on any of the following objects: the UCM project VOB, the component, the containing stream; and if you are importing a label type, the label type being imported.

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 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 154.

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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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@\pvob1**

SEE ALSO

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

mkcomp

Creates a UCM component object

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if there are locks on the following objects: the 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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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**

mkcomp

mkfolder

Creates a folder for a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if one or more of these objects are locked: UCM project VOB.

Mastership: (Replicated VOBs only) No mastership restrictions.

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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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**

mkfolder

mkproject

Create a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if any of the following objects are locked: the 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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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**

mkproject

mkstream

Creates a stream for a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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@\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**

mktrigger

Attaches a trigger to an element or UCM object

APPLICABILITY

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

Platform
UNIX
Windows

SYNOPSIS

- ClearCase and ClearCase LT only—Attach a trigger to an element or a UCM object:

  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* | *ucm-object-selector* } ...

- Attache only—Attach a trigger to an element:

  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

The **mktrigger** command attaches a trigger to one or more elements or UCM objects. An attached trigger fires (executes the trigger action) when the element (or any of its versions) or the UCM object 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.

NOTE: A trigger type object, created with **mktrtype –element** must already exist in the VOBs containing the specified elements. Similarly, you use **mktrtype –ucmobject** to create a trigger type object in the project VOB containing the specified UCM objects before you can use this command.

Element 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.)

RESTRICTIONS

Identities: For each object processed, you must be one of the following: object group member, object owner, VOB owner (for an element trigger), project VOB owner (for a UCM object trigger), or:

- UNIX: **root**
- ClearCase on Windows: member of the ClearCase group
- ClearCase LT on Windows: local administrator of the ClearCase LT server host

See the **permissions** reference page.

Locks: An error occurs if any of the following objects are locked: VOB (for an element trigger), project VOB (for a UCM object trigger), object type, object, 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 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 ELEMENT 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). UNIX VOB symbolic links are not traversed during the recursive descent into the subtree.

CONTROLLING ELEMENT 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 TYPE RESTRICTIONS. *Default:* If *trigger-type-name* is defined with a restriction to one or more object types, **mktrigger** refuses to process an object of another type.

–f·orce
Attaches a trigger to an object whose type does not match the definition of the trigger type. Such a trigger does not fire unless you change the object'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 ELEMENT. *Default:* None.

pname ...
One or more pathnames, specifying elements to which the specified trigger type is to be attached.

SPECIFYING THE UCM OBJECT. *Default:* None.

ucm-object-selector ...
The name of the UCM object. Specify *ucm-object-selector* in the form [*ucm-object-type***:**]*type-name*[**@***vob-selector*].

ucm-object-type	Name of the UCM object type
vob-selector	UCM project VOB specifier
	Specify *vob-selector* in the form [**vob:**]*pname-in-vob*
pname-in-vob	Pathname of the project VOB-tag (whether or not the project VOB is mounted) or of any file-system object within the project VOB (if the project VOB is mounted)

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

mktrigger

mktrtype

Creates a trigger type object

APPLICABILITY

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

Platform
UNIX
Windows

SYNOPSIS

- ClearCase, ClearCase LT, and Attache only—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 ...

- ClearCase, ClearCase LT, and Attache only—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 ...

- ClearCase and ClearCase LT only—Create a UCM trigger type:

  mktrtype –ucm·object [**–a·ll**] [**–rep·lace**]
  { **–pre·op** | **–pos·top** } *opkind*[,...] [**–nus·ers** *login-name*[,...]]
  { **–exe·c** *command*
  | **–execu·nix** *command*
  | **–execw·in** *command*
  | **–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 ...

- A *restriction-list* for an element trigger type 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: **–***xxx***type** *aaa*,*bbb* is equivalent to **–***xxx***type** *aaa* **–xxtype** *bbb*.

- A *restriction-list* for a UCM trigger type contains one or more of:

  –com·ponent *component-selector*[,...] (*Default:* All components)
  –pro·ject *project-selector*[,...] (*Default:* All projects)
  –str·eam *stream-selector*[,...] (*Default:* All streams)

- An *inclusion-list* for an element trigger type 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: **–***xxx***type** *aaa*,*bbb* is equivalent to **–***xxx***type** *aaa* **–xxtype** *bbb*.

DESCRIPTION

The **mktrtype** command creates one or more trigger types for use within a VOB or UCM project 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—that is, 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.

The trigger types are as follows:

- 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 this type, 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.

- A UCM trigger type is attached to one or more UCM objects, such as a stream or activity, and fires when the specified operation is invoked on the UCM object. You can also create an all-UCM-object trigger type. Like the all-element type, this type is implicitly attached to all existing and potential UCM objects in the project VOB (that is, no **mktrigger** command is required).

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 or command prompt. 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 under the identity of the process that caused the trigger to fire.

Interactive Trigger Action Scripts

A script or batch file 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 in 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 or writes a comment to a file. You can use the **CLEARCASE_PPID** environment variable to help synchronize multiple firings (for more information, see *Trigger Environment Variables*).

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 identities. Firing of a 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 UCM project VOB, or outside them—the occurrence of the operation. If a postoperation trigger action returns a nonzero exit status, a `failed exit status` warning message is printed, but other trigger actions, if any, are executed.

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 a 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 or UCM 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 192*.

RESTRICTIONS

Identities: For each object processed, you must be one of the following: type owner (applies to **–replace** only), VOB owner (element trigger types), project VOB owner (UCM trigger types) or:

- UNIX: **root**
- ClearCase on Windows: member of the ClearCase group
- ClearCase LT on Windows: local administrator of the ClearCase LT server host

See the **permissions** reference page.

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 implicitly attached to all VOB objects, subject to the restriction list.

–ucm·object
Creates a UCM object trigger type, which can be attached to individual UCM objects with **mktrigger**.

–ucm·object –a·ll
Creates an all-UCM-object trigger type, which is implicitly attached to all project VOB objects, subject to the restriction list.

–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**.

If an instance of an element or UCM trigger type is currently attached to any object, the replacement definition must correspond in kind: the new definition must be of an element trigger type or a UCM trigger type (but not an all-element or all-UCM object 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.

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.

–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.

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 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.

ClearCase, ClearCase LT, and Attache on Windows—If you do not run **mktrtype** from the **cleartool** prompt, enclose *command*—and any single quotes—in double quotes (" ' *command* ' "). (See also the **cleartool** reference page.)

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:

–exec 'cmd /c copy %CLEARCASE_PN% %HOME%'

–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 TO UNIX USERS: 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 element 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 **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 **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 **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 **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 or other programs 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**.

UCM TRIGGER TYPES: SPECIFYING A RESTRICTION LIST: *Default:* For **–component**, all components; for **–project**, all projects; for **–stream**, all streams.

–com·ponent *component-selector*[,...]
–pro·ject *project-selector*[,...]
–str·eam *stream-selector*[,...]
Use one or more of the above options to specify a set of UCM objects for the restriction list. At trigger firing time, the items on the restriction list form a logical condition: if the condition is met, the trigger fires.

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

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

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

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 the environment variable **CLEARCASE_TRACE_TRIGGERS** is set to a nonnull value for the process that causes the trigger to fire, a message that includes the trigger type name is printed 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 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 or UCM project 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 **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 or project VOB is mounted) or of any file-system object within the VOB or project 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**). In UNIX, the operation fires a trigger only if the affected object is a type object specified on the inclusion list, which 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 7 lists the operation keywords (*opkind*) for use in definitions of element and all-element trigger types (**–element** and **–element –all**). For any *opkind*, not all restrictions specified in the *restriction-list* argument are especially relevant: Table 7 also shows which restrictions are checked for each *opkind*. The *opkind*s in capitals (such as **MODIFY_ELEM**) specify all *opkind*s that appear under them; in other words, they are generalizations of the more specific *opkind*s.

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 7 Element 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	

Table 7 Element Trigger Definition Operation Keywords

Operation Keyword	Restrictions Checked when Trigger Fires
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	
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

Table 7 Element Trigger Definition Operation Keywords

Operation Keyword	Restrictions Checked when Trigger Fires
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 Operations for UCM Objects and All-UCM-Object Trigger Types

Table 8 lists the operation keywords (*opkind*) for use in definitions of UCM object and all-UCM-object trigger types (**–ucmobject** and **–ucmobject –all**). The table shows the kind of UCM object to which the trigger may be attached—you may also use **–all** to specify all UCM objects. For any UCM operation, not all restrictions specified in the *restriction-list* argument are especially relevant: Table 8 also shows which restrictions are checked for each operation. You can use the **UCM** operation as a synonym for all other UCM operations; it causes a trigger to fire when any UCM operation for which triggers are enabled occurs.

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.

Table 8 UCM Object Trigger Definition Operation Keywords

Operation Keyword	Object Type	Restrictions Checked when Trigger Fires
UCM		
deliver_start	Target (integration) stream	Stream, Project
deliver_cancel	Target (integration) stream	Stream, Project

Table 8 UCM Object Trigger Definition Operation Keywords

Operation Keyword	Object Type	Restrictions Checked when Trigger Fires
deliver_complete	Target (integration) stream	Stream, Project
rebase_start	Target (development) stream	Stream, Project
rebase_cancel	Target (development) stream	Stream, Project
rebase_complete	Target (development) stream	Stream, Project
mkactivity	Stream that is to contain the activity	Stream, Project
setactivity	Activity being set	Stream, Project
mkstream	Project that is to contain the stream	Project
mkbl	Component that is to contain the baseline	Stream, Component, Project. No triggers are fired if the baseline is initial; if imported, triggers fire but the environment variables **CLEARCASE_STREAM** and **CLEARCASE_PROJECT** are undefined.

Trigger Environment Variables

The following list shows the EVs that are set in the environment in which a trigger action script 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_ACTIVITY
(All **deliver** and **rebase** operations; **checkin**, **checkout**, **mkactivity**, **setactivity**, **uncheckout**) The UCM activity, if applicable, involved in the operation that caused the trigger to fire. For **checkin**, **checkout** and **uncheckout** operations, the activity that is set in the view used for the operation. For the **mkactivity**, **deliver_start**, and **rebase_start** operations, this environment variable is set only for a post-op trigger.

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_BASELINES
(All **rebase** operations, **mkbl**) A space-separated list of all UCM foundation baselines to which the destination stream is to be rebased. For the **mkbl** operation, a post-op trigger only (list of length 1); for the **chbl** and **rmbl** operations, the list may specify only 1 foundation baseline.

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_COMPONENT
(**mkbl**) The UCM component containing the object involved in the action that caused the trigger to fire, if applicable.

CLEARCASE_DLVR_ACTS
(**deliver_start**, **deliver_complete**) A space-separated list of all UCM activities merged during the **deliver** operation.

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) "/." or "\." pathname component:
  `/vobs/proj/./releasedir` *(if VOB is mounted at '/vobs/proj')*
  `\proj1\.\releasedir` *(if VOB-tag is \proj1)*

- With an all-element trigger, a pathname in the root directory of a VOB is reported with an extra (but still correct) "/." or "\." pathname component:
- Some **cleartool** and Attache commands rename files during their execution. Usually, such manipulations are unnoticeable, but you may need to adjust your trigger scripts or batch files accordingly. For example, the script for a preoperation **mkelem** trigger may need to operate on file name "$**CLEARCASE_PN**.mkelem" instead of "$**CLEARCASE_PN**" (UNIX) or on name "%**CLEARCASE_PN**%.mkelem" instead of "%**CLEARCASE_PN**%" (Windows)
- 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**, **deliver**, **rebase**) 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**
move | mv	**lnname**	**rmname**
	rmname	**lnname**
deliver_start	**mkactivity**	**deliver_start**
	setactivity	
	mkbl	
rebase_start	**mkactivity**	**rebase_start**
	setactivity	
	mkbl	

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_PROJECT
(All **deliver** and **rebase** operations; **mkactivity**, **mkstream**, **mkbl**, **setactivity**) The UCM project containing the object involved in the action that caused the trigger to fire, if applicable. Not set for the **mkbl** operation if this is an initial (or imported) baseline.

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_SLNKTXT
(**mkslink**; that is, the **ln –s** command) Text of the new VOB symbolic link.

CLEARCASE_SNAPSHOT_PN
(All operations executed in a snapshot view) The path to the root of the snapshot view directory in which the operation that caused the trigger to fire took place.

CLEARCASE_STREAM
(All **deliver** and **rebase** operations; **mkactivity**, **setactivity**, **mkstream**, **mkbl**) The UCM stream containing the object involved in the action that caused the trigger to fire, if applicable. For the **mkstream** operation, a post-op trigger only. Not set for the **mkbl** operation if this is an initial (or imported) baseline.

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 real user ID or the Windows user-ID.

CLEARCASE_VAL
(**mkattr**) String representation of attribute value for **CLEARCASE_ATTYPE** (for example, `"Yes"` or `4657`).

CLEARCASE_VIEW_KIND
(All operations) The kind of view in which the operation that caused the trigger to fire took place; the value may be **dynamic**, **snapshot**, or **snapshot web**.

CLEARCASE_VIEW_TAG
(All non-UCM operations; for UCM, all **deliver** and **rebase** operations and **setactivity**) 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 or UCM project 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: Trigger environment variables are typically evaluated when the trigger fires, not when you enter the **mktrtype** command. If this is the case, escape the **$** (UNIX) or **%** (Windows) environment variable symbol according to the conventions of the shell you are using. 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 newly created 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 ^**
'M:\%CLEARCASE_VIEW_TAG%\%CLEARCASE_VOB_PN%\trigs\check_ext %CLEARCASE_PN%' ^
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 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 ^**
'M:\%CLEARCASE_VIEW_TAG%\%CLEARCASE_VOB_PN%\trigs\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

- Create a preoperation trigger type that fires on the **deliver_start** operation.

  cmd-context **mktrtype -ucmobject -all -preop deliver_start $PREOPCMDU $PREOPCMDW -stream $STREAM -nc $PREOPTRTYPE**

- Create a post-operation trigger type that fires on the **deliver_complete** operation.

  cmd-context **mktrtype -ucmobject -all -postop deliver_complete $WCMD $UCMD -stream $STREAM -nc $TRTYPE**

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

Platform
UNIX
Windows

SYNOPSIS

- ClearCase and Attache on UNIX 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 on Windows 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*] [**–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*]
  | **–vws** *view-storage-pname* [**–hos·t** *hostname* **–hpa·th** *host-storage-pname*
  –gpa·th *global-storage-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. 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. In ClearCase LT, the locations of view storage directories are restricted to the ClearCase LT server host.
- Creates a view-tag, the name by which users access a dynamic view. Snapshot views also have view-tags, but these are for administrative purposes; 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**. This directory is distinct from the view storage directory.
- 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.

DISCONNECTED USE OF SNAPSHOT VIEWS

If you want to use a snapshot view when disconnected from the network:

- Create the snapshot view directory on the device that is to be disconnected from the network from time to time.

- Create the view storage directory in a location that is consistently connected to the network, on a host where ClearCase or ClearCase LT has been installed. This location could be a server storage location (specified by **–stgloc**) or a location specified by the **–vws** option. Do not use **–colocated_server**: this option creates the view storage directory as a subdirectory of the snapshot view directory (where, of course, it will be subject to disconnection from the network).

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.

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.

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 directory, you must back up recursively both the view storage directory and the snapshot view 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 an operating system file deletion command. See the **rmview** reference page for details.

INFORMATION SPECIFIC TO PRODUCTS, VIEW TYPES AND PLATFORMS

This section contains information about view creation that differs depending on the product, view type, and platform you are using.

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** or **omake** in this kind of view, **clearmake** or **omake** 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 on UNIX 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.

ClearCase and Attache Dynamic Views on UNIX 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 and Attache Dynamic Views on Windows 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 .dynamic-views root directory (by default, **M**:\). (For example, specifying **–tag gamma** creates the entry **gamma**.)

After a dynamic view is activated, you can assign it to a drive letter with the **net use** command or by clicking **Tools→Map Network Drive** in Windows Explorer; 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.

ClearCase, Attache, and ClearCase LT on UNIX Only—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.

RESTRICTIONS

Identities: No special identity 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 the viewroot directory. For example, the view **experiment** appears as **/view/experiment** (UNIX) or **M:\experiment** (Windows).

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 *dynamic-view-storage-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 the location specified by *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 *ClearCase, Attache, and ClearCase LT on UNIX Only—View Creator Identity and umask Permissions*.

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 heterogeneous 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.

  For information on using this option to create snapshot views for disconnected use, see the section, *DISCONNECTED USE OF SNAPSHOT VIEWS*.

  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 a view storage directory that is colocated with the snapshot view directory; specifically, the view storage directory is created as a subdirectory of the snapshot view directory (*snapshot-view-pname*).

  We recommend you use **–stgloc** rather than this option whenever possible.

–vws
: Specifies the location for the snapshot view storage directory. On Windows systems, this must be a UNC name.

  For information on using this option to create snapshot views for disconnected use, see the section, *DISCONNECTED USE OF SNAPSHOT VIEWS*.

  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: The argument names shown above are generalizations of the argument names as they appear in the synopses for this command in association with the **–colocated_server** and **–vws** options.

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 the snapshot view directory (*snapshot-view-pname*), not the colocated view storage directory.

When you use one or more of the **–host**/**–hpath**/**–gpath** options in combination with **–vws**, the values you specify for **–host**/**–hpath**/**–gpath** must correspond to the view storage directory (*view-storage-pname*), not the snapshot view 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 under the dynamic views root directory (on UNIX, this directory is **/view**; on Windows, **M:**)
- *dynamic-view-storage-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.

In addition, on Windows systems:

- *dynamic-view-storage-pname* must be a UNC name
- The directory must not be within a Windows special share, such as the share that is designated by *driveletter***$** and that allows administrators to access the drive over the network.

snapshot-view-pname

The location at which the snapshot view directory is to be created.(An error occurs if something already exists at this pathname.) You can create a snapshot view directory at any location in the file system where operating system permissions allow you to create a subdirectory, with the restriction that you cannot create a snapshot view under the dynamic views root directory (on UNIX, this directory is **/view**; on Windows, **M:**).

In addition, on Windows systems:

- *snapshot-view-pname* must be a UNC name if and only if the storage is colocated (colocated storage can be the default in the circumstances described previously).
- For a colocated server, the snapshot view directory must not be within a Windows special share, such as the share that is designated by *driveletter***$** and that allows administrators to access the drive over the network.

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. In **cleartool** interactive mode, *cmd-context* represents the interactive **cleartool** prompt. In Attache, *cmd-context* represents the workspace prompt.

NOTE: In the UNIX 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.

On a UNIX system, 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
```

- On a Windows systems, create a dynamic view and assign it the view-tag **main_r2**. This example assumes that host **pluto** shares its C: drive via sharename **c_share**.

  ```
  cmd-context  mkview -tag main_r2 \\pluto\c_share\vw_store\winproj\main_r2.vws

  Created view.
  Host: pluto
  Local path: c:\vw_store\winproj\main_r2.vws
  Global path: \\pluto\c_share\vw_store\winproj\main_r2.vws
  It has the following rights:
  User : anne : rwx
  Group: dev : rwx
  Other: : r-x
  ```

- On a UNIX system, 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
  ```

- On a UNIX system, 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
```

- On a UNIX system, reate 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.

- On a UNIX system, 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
```

- On a Windows system, create a dynamic view, assign it the view-tag **smg_bigvw**, and specify a large cache size.

  cmd-context **mkview –tag smg_bigvw –cachesize 1m \\neon\vws\smg_bigvw.vws**

```
Created view.
Host-local path: neon:C:\USERS\vws\smg_bigvw.vws
Global path:      \\neon\vws\smg_bigvw.vws
```

- On a UNIX system, 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".
  ```

- On a UNIX system, 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".
  ```

- On a UNIX system, 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**

omake

ClearCase build utility — maintain, update, and regenerate groups of programs

APPLICABILITY

Product	Command Type
ClearCase	command

Platform
Windows

SYNOPSIS

omake [**–f** *makefile* ...] [**–b** *builtins-file* ...]
[**–akinservdphzACDGM**] [**–x** *file*] [**-OLWT**]
[**-EN | -EP | -EO**] [**-#1**] [**-#2**] [**-#4**] [**-#8**]
[*macro*=*value* ...] [*target_name* ...]

DESCRIPTION

omake is a ClearCase utility for making (building) software. It includes many of the configuration management (CM) facilities provided by the **clearmake** utility. It also features emulation modes, which enable you to use **omake** with makefiles that were constructed for use with other popular **make** variants, including Microsoft NMAKE, Borland Make and the PVCS Configuration Builder (Polymake).

NOTE: **omake** is intended for use in dynamic views. You can use **omake** in a snapshot view, but none of the features that distinguish it from ordinary **make** programs — build avoidance, build auditing, derived object sharing, and so on — works in snapshot views. The rest of the information in this reference page assumes you are using **omake** in a dynamic view.

omake features a number of ClearCase extensions:

- **Configuration Lookup** — a build-avoidance scheme that is more sophisticated than the standard scheme based on the time-modified stamps of built objects. For example, this guarantees correct build behavior as C-language header files change, even if the header files are not listed as dependencies in the makefile.
- **Derived Object Sharing** — developers working in different views can share the files created by **omake** builds.

- **Creation of Configuration Records** — software bill-of-materials records that fully document a build and support rebuildability; also includes automatic dependency detection.

Related Reference Pages

The following reference pages include information related to **omake** operations and results:

clearmake	Alternative make utility - provides the same functionality as the **clearmake** tool in the UNIX version of ClearCase.
clearaudit	Alternative to make utilities, for performing audited builds without makefiles.
lsdo	**cleartool** subcommand to list derived objects created by **omake** or **clearaudit**.
catcr, **diffcr**	**cleartool** subcommands to display and compare configuration records created by **omake** or **clearaudit**.
rmdo	**cleartool** subcommand to remove a derived object from a VOB.

See also *Building Software with ClearCase*.

View Context Required

For a build that uses the data in one or more VOBs, the command interpreter from which you invoke **omake** must have a view context—you must be on a drive assigned to a view or the dynamic-views drive (default: **M:**\). If you want derived objects to be shared among views, you should be on a drive assigned to a view.

You can build objects in a standard directory, without a view context, but this disables many of **omake**'s special features.

omake AND MAKEFILES

omake is designed to read makefiles in a way that is compatible with other **make** variants. For details, see the *ClearCase OMAKE Manual*.

HOW BUILDS WORK

In many ways, ClearCase builds adhere closely to the standard **make** paradigm:

1. You invoke **omake**, optionally specifying the names of one or more targets. (Such explicitly-specified targets are termed goal targets.)
2. **omake** reads zero or more makefiles each of which contains targets and their associated build scripts.
3. **omake** supplements the makefile-based software build instructions with its own built-in rules. (And when it runs in a compatibility mode, **omake** also defines built-in rules specific to that mode.)

4. For each target, **omake** performs build avoidance, determining whether it actually needs to execute the associated build script ("perform a target rebuild"). It takes into account both source dependencies ("have any changes occurred in source files used in building the target?") and build dependencies ("must other targets be updated before this one?").

   The sources can be on the dependency list, or may be detected by **omake**. A source is a target or file that must exist and be up-to-date before the target is built. The dependency list is used to make decisions about build ordering (which targets need to be built and in which order). Detected dependencies (source dependencies detected automatically by **omake**) are also used to determine if a DO can be reused or is out of date.

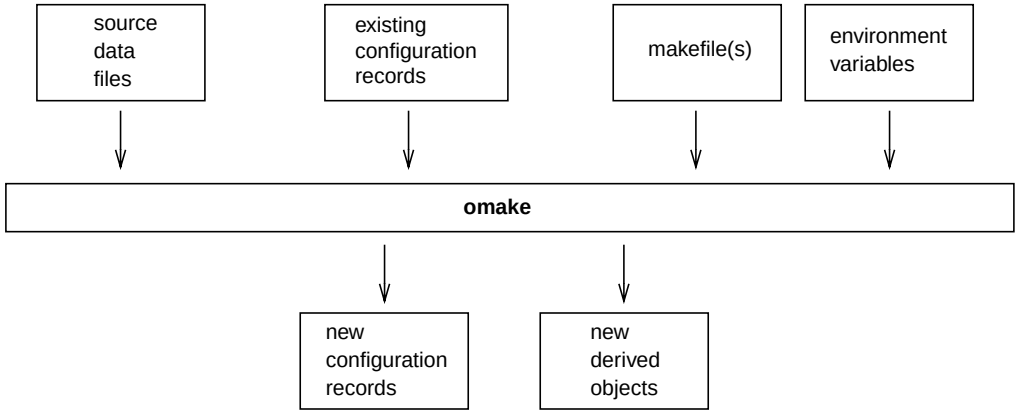
   Like detected dependencies, predefined dependencies are used for DO reuse questions, but they are also used for determining which files should be built and when they should be built. For example, for a **.exe** file, you must predefine all the **.obj** files to ensure that they are built first; for an **.obj** file, you list the **.c** or **.cpp** files, but header files can be left off and omake still detects them.

   The difference is very important for a first build, when there are no existing DOs and only the makefile exists to define the dependencies and what target depends on what other target.

5. If it decides to perform a target rebuild, **omake** executes its build script.

The following sections describe special **omake** build features in more detail. Figure 8 illustrates the associated data flow.

Figure 8 Data Flow in an **omake** Build



CONFIGURATION RECORDS AND DERIVED OBJECTS

In conjunction with the MVFS file system, **omake** audits the execution of all build scripts, keeping track of file usage at the OS-system-call level. For each execution of a build script, it

creates a configuration record (CR), which includes the versions of files and directories used in the build, the build script, build options, (for example, macro assignments) and other related information. A copy of the CR is stored in the VOB database of each VOB in which the script has built new objects.

A file created within a VOB by a build script is called a derived object (DO), and it can be shareable or nonshareable. When a shareable derived object is built in a view, a corresponding VOB database object is also created. This enables any view to access and possibly share (subject to access permissions) any derived object, no matter what view it was originally created in. When a build tool creates a nonshareable derived object, the tool does not write any information about the DO to the VOB. Therefore, the DO is invisible to other views and cannot be winked in by them. Builds that create nonshareable DOs are called express builds. For more information about using express builds, see *Preventing Winkin to Other Views*.

For each build script execution, ClearCase logically associates each DO that was created in that execution with the build script's CR.

You can suppress the creation of CRs and derived objects with the **–L** option and ClearCase-specific directives. See *Building Software with ClearCase* for details on CRs and derived objects, and see the *ClearCase OMAKE Manual* for information on ClearCase-specific directives.

(Files created in non-VOB directories are not derived objects — see the *MVFS FILES AND OBJECTS OUTSIDE THE MVFS* section.)

Configuration Record Hierarchies

A typical makefile has a hierarchical structure. Thus, a single invocation of **omake** to build a high-level target can cause multiple build scripts to be executed and, accordingly, multiple CRs to be created.

CONFIGURATION LOOKUP AND WINKIN

For directory targets, **omake** uses standard **make** logic.

When a target names a non-directory file in a VOB, **omake** (by default) uses configuration lookup to determine whether a build is required. This involves a comparison of the CRs of existing derived objects with the current build configuration:

- the versions of elements selected by the view's config spec
- the build options to be applied, as specified on the **omake** command line, in the environment, or in the makefile(s)
- the build script to be executed

In performing configuration lookup, **omake** considers a DO version (a derived object that has been checked in as a version of an element) only if the version was created in place. That is, if you

copy a DO to a different location from where it was created and check it in there, **omake** will not consider the DO version.

omake first tries to avoid rebuilding by *reusing* a DO in the current view; this succeeds only if the CR of the candidate DO matches the current build configuration. For the purpose of rebuilding, a *branch*\0 version of a file selected by a view is considered to match its non-zero predecessor version in a CR.

omake can also avoid rebuilding by finding another DO, built in another view, whose CR matches the current build configuration. In this case, it will wink in that derived object, causing it to be *shared* among views. Other derived objects created by the same build script (termed siblings) are winked in at the same time. **omake** rebuilds a target only if it is unable to locate any existing derived object that matches the current build configuration.

DO versions must be checked out before they can be re-used or winked in. The **-C** option to **omake** provides support for automatically checking out these DOs before they are used. The **CCASE_AUTO_DO_CI** environment variable provides a means to automatically check in DOs checked out by **omake -C.** Checkouts executed by this feature behave like any **cleartool checkout** does with respect to reservation. Methods that can be used to change **cleartool checkout**'s default reservation policy apply here as well. The checkouts are not audited. Checkins preserve the timestamp of the DO as though **cleartool checkin -ptime** were used. This feature is fully compatible with **checkout** or **checkin** triggers, which fire normally when the event occurs.

NOTE: Certain special targets may prevent winkin even if the build configuration conditions are exactly the same. For example, if you are using **.pdb** files in Visual C++, winkin of any target that has a **.pdb** for a sibling will not occur, even though all versions of dependencies in the config record are selected by the view in which the build occurs.

The .cmake.state File

The **.cmake.state** file is a view-private cache of config records for derived objects built in the view during a particular build. **omake** creates this file in the directory that was current when the build started. During subsequent builds in the view, **omake** references the file instead of communicating with the VOB. This makes configuration lookup faster, improving **omake** performance.

You can delete **.cmake.state** files if they get too large. If **omake** looks for a **.cmake.state** file and it doesn't exist, no errors occur and **omake** creates a new file.

Suppressing Configuration Lookup

You can override the default configuration lookup behavior with command options and ClearCase-specific directives (see the *ClearCase OMAKE Manual* for information on these directives). For example, **–L** turns off configuration lookup, basing rebuild decisions on time-modified stamps, and **–W** disables winkin of DOs from other views.

Preventing Winkin to Other Views

You can prevent derived objects that you create from being winked in to other views. For more information, see *Working with Derived Objects and Configuration Records* in *Building Software with ClearCase*.

CACHING UNAVAILABLE VIEWS

When **omake** shops for a derived object to wink in to a build, it may find DOs from a view that is unavailable (because the view server host is down, the **albd_server** is not running on the server host, and so on). Attempting to fetch the DO's configuration record from the view causes a long time-out, and the build may attempt to contact the same view multiple times.

omake maintains a cache of tags of inaccessible views. For each view-tag, **omake** records the time of the first unsuccessful contact. Before trying to access a view, **omake** checks the cache. If the view's tag is not listed in the cache, **omake** tries to contact the view. If the view's tag is listed in the cache, **omake** compares the time elapsed since the last attempt with the time-out period specified by the **CCASE_DNVW_RETRY** environment variable. If the elapsed time is greater than the time-out period, **omake** removes the view-tag from the cache and tries to contact the view again.

NOTE: The cache is not persistent across **omake** sessions. Each recursive or individual invocation of **omake** attempts to contact a view whose tag may have been cached in a previous invocation.

The default time-out period is 60 minutes. To specify a different timeout period, set **CCASE_DNVW_RETRY** to another integer value (representing minutes). To disable the cache, set **CCASE_DNVW_RETRY** to 0.

MVFS FILES AND OBJECTS OUTSIDE THE MVFS

All files with pathnames below a VOB-tag (VOB mount point) are termed MVFS files:

- checked-in versions of file elements (data stored in VOB)
- checked-out versions of file elements (data stored in view)
- other view-private files
- derived objects

Conversely, a non-MVFS object is any file or directory whose pathname is not under a VOB-tag; such objects are not version controlled. By default, non-MVFS objects are not audited during **omake** builds.

OPTIONS AND ARGUMENTS

omake supports the options below. In general, standard **make** options are lowercase characters; **omake** extensions are uppercase. Options that do not take arguments can be ganged on the command line (for example, **–rOi**).

–f *makefile*

Use *makefile* as the input file. If you omit this option, **omake** looks for input files named *makefile* and *Makefile* (in that order) in the current working directory. You can use more than one **–f** *makefile* argument pair. Multiple input files are effectively concatenated.

–b *file*

Specify an initialization (built-ins) file to be read instead of the default. If *file* is the empty string, **omake** does not read an initialization file. Valid empty strings are **"–b "** (one space), **–b" "**, or **–b ""**.

NOTE: If you do not include the **–b** option, **omake** uses the file named by the OMAKECFG environment variable. If this environment variable is not set, **omake** looks for a file called **make.ini** in (in order) the current directory, *ccase-home-dir***bin**, and in directories specified by the INIT environment variable.

–a

Rebuild all goal targets specified on the command line, along with the recursive closure of their dependencies, regardless of whether or not they need to be rebuilt.

–k

Abandon work on the current entry if it fails, but continue on other targets that do not depend on that entry.

–i

Ignore error codes returned by commands.

–n

(no-execute) List command lines from the makefile for targets which need to be rebuilt, but do not execute them. Even lines beginning with an at-sign (@) character are listed.

To override this option for a recursive make, use the .MAKE target attribute. For example:

```
nt .MAKE :
  cd nt.dir & $(MAKE) $(MFLAGS)
```

Typing the command **omake –n nt** does a `cd nt.dir`, then a recursive make with `omake –n`. Without the .MAKE attribute, **omake** would display but not execute the **(cd nt.dir & $(MAKE) $(MFLAGS)** line.

–s

(silent) Do not list command lines before executing them.

–e

Environment variables override macro assignments within the makefile. (But *macro*=*value* assignments on the command line override environment variables.)

–r

Do not use the built-in rules.

–v

(verbose) Slightly more verbose than the default output mode. Particularly useful features of verbose mode include:

- listing of why **omake** does not reuse a DO that already appears in your view (for example, because its CR does not match your build configuration, or because your view does not have a DO at that pathname)
- listing of the names of DOs being created

–d

(debug) Quite verbose; appropriate only for debugging makefiles.

–p

Lists all target descriptions and all macro definitions, including target-specific macro definitions and implicit rules.

–h

Displays the command-line syntax.

–x *file*

Redirects error messages into *file*. If *file* is **"-"**, the error messages are redirected to standard output.

–z

Ignore the MFLAGS macro.

–A

Use automatic dependencies. This option is enabled only if you are not using configuration lookup (because you are processing non-MVFS files or using the **–W** option).

–C

(Check out DOs) Before building or winking in a target, **omake** determines whether the target is a checked-in DO visible in the view at the path named in the makefile. If such a DO is found, **omake -C** checks it out before rebuilding it or winking it in.

–D

Keep-directory mode. The first access of a directory to look for a file results in the directory being read into memory.

–G

Restricts dependency checking to makefile dependencies only — those dependencies declared explicitly in the makefile or inferred from an inference rule. All detected dependencies are ignored. For safety, this automatically disables winkin of DOs from

other views; it is quite likely that other views select different versions of detected dependencies.

For example, a derived object in your view may be reused even if it was built with a different version of a header file than is currently selected by your view. This option is mutually exclusive with **–W**.

–M

Makes the makefile before reading it.

–EN

Emulates Microsoft NMAKE utility.

–EP

Emulates PVCS Configuration Builder (PolyMake) utility.

–EO

Default emulation mode (that is, no emulation).

For details on emulation features, see the *ClearCase OMAKE Manual*.

–O
–L (mutually exclusive)

–O compares only the names and versions of objects listed in the targets' CRs; it does not compare build scripts or build options. This is useful when this extra level of checking would force a rebuild that you do not want. Examples:

- The only change from the previous build is the setting or canceling of a "compile-for-debugging" option.
- A target was built using a makefile in the current working directory. Now, you want to reuse it in a build to be performed in the parent directory, where a different makefile builds the target (with a different script, which typically references the target using a different pathname).

–L makes rebuild decisions using the standard algorithm, based on time-modified stamps; configuration lookup is disabled. Also suppresses creation of configuration records. All MVFS files created during the build will be view-private files, not derived objects.

–W

Restricts configuration lookup to the current view only. Winkin of DOs from other views is disabled.

–T

Examines sibling derived objects (objects created by the same build rule that created the target) when determining whether a target object in a VOB can be reused (is up to date). By default, when determining whether a target can be reused, **omake** ignores

modifications to sibling derived objects. **–T** directs **omake** to consider a target out of date if its siblings have been modified or deleted.

–#1

Read-time debugging mode. Displays **omake** reading makefiles and interpreting conditional directives.

–#2

Displays a warning when **omake** tries to expand the value of an undefined macro.

–#4

Displays a warning when **omake** reads a makefile line that it can't understand.

–#8

Do not delete generated response files and batch files.

MAKE MACROS AND ENVIRONMENT VARIABLES

String-valued variables called make macros can be used anywhere in a makefile: in target lists, in dependency lists, and/or in build scripts. For example, the value of make macro CFLAGS can be incorporated into a build script as follows:

cl $(CFLAGS) msg.c

Conflict Resolution

Conflicts can occur in specifications of make macros and environment variables. For example, the same make macro might be specified both in a makefile and on the command line; or the same name might be specified both as a make macro and as an environment variable.

omake resolves such conflicts similarly to other **make** variants:

- Make macros specified on the command line override any other settings.
- Make macros specified in a makefile or **make.ini** file have the next highest priority.
- Builtin macros override EVs, which in turn have the lowest priority.

Using the **–e** option changes the precedence rules — EVs get higher priority than make macros specified in a makefile.

CONFLICT RESOLUTION DETAILS. The following discussion treats this topic more precisely (but less concisely).

omake starts by converting all EVs in its environment to make macros. These EVs will also be placed in the environment of the command interpreter process in which a build script executes. Then, it adds in the make macros declared in the makefile. If this produces name conflicts, they are resolved as follows:

- If **omake** was not invoked with the **–e** option, the make macro wins: the macro value overwrites the EV value in the environment.
- If **omake** was invoked with the **–e** option, the EV wins: the EV value becomes the value of the make macro.

Finally, **omake** adds make macros specified on the command line; these settings are also added to the environment. These assignments *always* override any others that conflict.

omake reads the following environment variable at startup:

CCASE_AUDIT_TMPDIR (or **CLEARCASE_BLD_AUDIT_TMPDIR**)
: Sets the directory where **omake** creates temporary build audit files. If this variable is not set, **omake** creates these files in **%tmp%**. All temporary files are deleted when **omake** exits. **CCASE_AUDIT_TMPDIR** must not name a directory under a VOB-tag; if it does, **omake** prints an error message and exits.

CCASE_AUTO_DO_CI
: Checks in DOs checked out by **omake –C** unless the build of the corresponding target fails or the automatic checkout of the DO or a sibling DO fails. Checkout comments are preserved. The **checkin** is invoked with the **-ptime** option to preserve the DO's modification time. This environment variable has no effect unless you specify **–C**.

  Default: Undefined

BUILD REFERENCE TIME AND BUILD SESSIONS

omake takes into account the fact that software builds are not instantaneous. As your build progresses, other developers can continue to work on their files, and may check in new versions of elements that your build uses. If your build takes an hour to complete, you would not want build scripts executed early in the build to use version 6 of a header file, and scripts executed later to use version 7 or 8. To prevent such inconsistencies, **omake** locks out any version that meets both these conditions:

- The version is selected by a config spec rule that includes the **LATEST** version label.
- The version was checked in after the time the build began (the build reference time).

This reference-time facility applies to checked-in versions of elements only; it does not lock out changes to checked-out versions, other view-private files, and non-MVFS objects. **omake** automatically adjusts for the fact that the system clocks on different hosts in a network may be somewhat out of sync (clock skew).

For more information, see *Pointers on Using ClearCase Build Tools* in *Building Software with ClearCase*.

EXIT STATUS

omake returns a zero exit status if all goal targets are successfully processed. It returns various nonzero exit status values when the build is not successful. See the *ClearCase OMAKE Manual*.

EXAMPLES

- Build target **hello.exe** without checking build scripts or build options during configuration lookup. Be moderately verbose in generating status messages.

  > **omake –v –O hello.exe**

- Build the default target in the default makefile, with a particular value of make macro **INCL_DIR**.

  > **omake INCL_DIR=c:\src\include_test**

- Build target **bgrs.exe**, restricting configuration lookup to the current view only. Have environment variables override makefile macro assignments.

  > **omake –e –W bgrs.exe**

- Unconditionally build the default target in a particular makefile, along with all its dependent targets.

  > **omake –a –f project.mk**

FILES

*ccase-home-dir***bin****builtins.cb**
*ccase-home-dir***bin****builtins.nm**
*ccase-home-dir***bin****make.ini**

SEE ALSO

Building Software with ClearCase, *ClearCase OMAKE Manual*, **clearmake**, **clearaudit**, **cleartool**, **config_spec**, **promote_server**, **scrubber**

rebase

Changes the configuration of a UCM stream

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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**]

rebase

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.

RESTRICTIONS

Identities: No special identity required.

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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rebase

rmactivity

Deletes a UCM activity

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: You must be the activity owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—local administrator of the ClearCase LT server host

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 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 specified 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rmactivity

rmbl

Removes a UCM baseline

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: You must be the baseline owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—the local administrator of the ClearCase LT server host

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 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 specified 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rmcomp

Removes a UCM component

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: You must be the component owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—the local administrator of the ClearCase LT server host

Locks: An error occurs if there are locks on any of the following objects: the component, the 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 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 specified 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rmcomp

rmfolder

Remove a UCM folder

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: You be the folder owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—the local administrator of the ClearCase LT server host

Locks: An error occurs if one or more of these objects are locked:the UCM project VOB, the folder.

Mastership: (Replicated VOBs only) Your current replica must master the folder.

Other: You cannot delete a folder if it contains any projects or other folders, or is the **RootFolder**.

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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rmfolder

rmproject

Removes a UCM project

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: You must be the project owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—the local administrator of the ClearCase LT server host

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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rmstream

Remove a UCM stream

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: You must be the stream owner, the project VOB owner, or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—the local administrator of the ClearCase LT server host

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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

rmstream

rmtrigger

Removes trigger from an element or UCM object

APPLICABILITY

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

Platform
UNIX
Windows

SYNOPSIS

- ClearCase and ClearCase LT only—Remove a trigger from an element or a UCM object:

  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* | *ucm-object-selector* } ...

- Attache only—Remove a trigger from an element:

  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 or UCM objects. The specified *trigger-type-selector* is not affected by **rmtrigger**. To delete the trigger type, use the **rmtype** command.

RESTRICTIONS

Identities: For each object processed, you must be one of the following: object group member, object owner, VOB owner (for an element trigger), project VOB owner (for a UCM object trigger) or

- UNIX—**root**
- ClearCase on Windows only—a member of the ClearCase group
- ClearCase LT on Windows only—the local administrator of the ClearCase LT server host

Locks: An error occurs if any of the following objects are locked: VOB (for an element trigger), project VOB (for a UCM object trigger), object type, object, 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 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). UNIX 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 ELEMENT. *Default:* None.

pname ...

One or more pathnames, specifying elements from which triggers (instances of the specified trigger type) are to be removed.

SPECIFYING THE UCM OBJECT. *Default:* None.

ucm-object-selector ...

The name of the UCM object. Specify *ucm-object-selector* in the form [*ucm-object-type***:**]*type-name*[**@***vob-selector*]

ucm-object-type Name of the UCM type

vob-selector UCM project VOB specifier

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

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

EXAMPLES

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

setactivity

Specifies the current UCM activity for your view

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

Locks: An error occurs if one or more of these objects are locked: UCM project VOB, the activity.

Mastership: (Replicated VOBs only) Your 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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

setactivity

setplevel

Changes the list of promotion levels in a UCM project VOB

APPLICABILITY

Product	Command Type
ClearCase	cleartool subcommand
ClearCase LT	cleartool subcommand

Platform
UNIX
Windows

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.

RESTRICTIONS

Identities: No special identity required.

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

Mastership: The current replica must master the project VOB.

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 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

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

The Windows examples that include wildcards or quoting are written for use in **cleartool** interactive mode. If you use **cleartool** single-command mode, you may need to change the wildcards and quoting to make your command interpreter process the command appropriately.

In **cleartool** single-command mode, *cmd-context* represents the UNIX shell or Windows command interpreter prompt, followed by the **cleartool** command. 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**

setplevel