Working in Base ClearCase

Contents

Working in Base ClearCase

Rational ClearCase Concepts 1

Recommended Reading Paths 1
To Access Online Information 1
ClearCase Elements and Activities 2
ClearCase Views 2
Snapshot Views and Dynamic Views 2
Versions, Elements, and VOBs 2
Selecting Elements and Versions 3
Config Specs for Snapshot Views 3
Config Specs for Dynamic Views 4
Criteria for Selecting Versions 4
Version Labels in Version-Extended Paths 5
Learning the Config Spec Syntax 6
View-Private Objects 6
The Base ClearCase-ClearQuest Integration 6
The Base ClearCase Schema and ClearQuest User Databases 6
ClearCase Triggers and Change Requests 7
Uses of the Base ClearCase-ClearQuest Integration 7
Parallel Development 8
Extended Namespace for Elements, Branches, and Versions 8

Setting Up a Work Environment 11

Choosing a Snapshot View or a Dynamic View 11
Choosing a Location and Name 12
Snapshot View: Choosing a Directory 12
Under the Hood: A Snapshot View Storage Directory 12
Locations for Snapshot View Storage Directories 13
Choosing a View Name 13
Using the View Tag 14
Dynamic View: Choosing a Location for the View Storage Directory 14
Choosing Locations for Dynamic View Storage Directories 14
Adjusting Your umask 15
The CCASE_BLD_UMASK Environment Variable 16

Creating the View with cleartool mkview 16
To Create a Snapshot View 16
Under the Hood: .ccase_svreg 17
To Create a Dynamic View 17
Under the Hood: The cleartool Command-Line Interface 17
Adding or Modifying Version-Selection Rules 18
To Copy or Include Version-Selection Rules 18
Snapshot View: Adding or Modifying Load Rules 19
Listing the VOB Namespace 19
VOB Namespace 19
To List the VOB Namespace 19
Adding or Modifying Load Rules 20
To Add or Modify Load Rules When Editing the Config Spec 20
To Add Load Rules with update –add_loadrules 21
Excluding Elements from Loading 21
To Exclude Elements 21
Under the Hood: VOB Links 22
Symbolic Links and Hard Links in Dynamic Views 22
Symbolic Links in Snapshot Views 22
Hard Links in Snapshot Views 23
Caution: Losing Data Because of VOB Hard Links 23
Setting Up ClearQuest User Database Defaults 24
To Control the Display Appearance 24
To Enable EMail Notification 24
To Set Defaults for Printing a Result Set 25
To Associate a Format and Record Type 25
To Change Result Set Print Appearance 26
To Set Headers and Footers 26
To Set Start-Up and Operational Defaults 26
To Run Queries at Startup 27
To Use the Query Wizard 27

Working in a View 29

Accessing Files 29
In a Snapshot View 29
Accessing Someone Else's Snapshot View 29
In a Dynamic View 29
To Set a Dynamic View 30
To Mount VOBs 30
Accessing Someone Else's Dynamic View 30

Checking Out Elements 30
To Check Out an Element 30
Reserved and Unreserved Checkouts 31
To Change the Status of a Checked-Out Version 32
Under the Hood: What Happens When You Check Out a File or Directory 33
Checking Out From a Snapshot View 33
Checking Out From a Dynamic View 34
Checking Out Elements in a VOB Enabled for ClearQuest 34
Logging On to a ClearQuest User Database 35
The Base ClearCase-ClearQuest Integration Interfaces 35
Using the Modified Graphic User Interface 35
Using the Modified Command Line Interface 36
Using the Menu to Associate a Checkout with a ClearQuest Entity 37
Working with Checkouts 38
Viewing the History of an Element 38
To View Element History 38
Comparing Versions of Elements 38
To Compare with a Predecessor 38
To Compare with a Version Other Than the Predecessor 39
To Compare with a Version in a Dynamic View 39
Tracking Checked-Out Versions 40
Prototype Builds 40
Canceling Checkouts 40
Under the Hood: Canceling Checkouts 41
Canceling Directory Checkouts 41
To Move and Delete Orphaned Elements 42
Checking In Files 42
To Check In Files 43
Merging the Latest Version with Your Changes 43
To Merge the Latest Version 44
Under the Hood: Checking In Files 45
Checking In From a Snapshot View 45
Checking In From a Dynamic View 45
Checking In Elements in a VOB Enabled for ClearQuest 45
Using the Association Dialog Window 46
Using the Command Line Interface Options 46
View the Versions for a Change Request from ClearQuest 46

Updating a Snapshot View 49

Starting an Update Operation 49

Updating the Entire View 49
To Start the Update Tool 50
Updating Files and Directory Trees 51
Under the Hood: What Happens When You Update a View 51
Unloading Elements 52
Unloading Files 52
Unloading Directories 53

Working On a Team 55

The Version Tree 55
Under the Hood: The Initial Version on a Subbranch 56
Working on Branches 58
The Version-Extended Path 58
Merging 60
Under the Hood: How ClearCase Merges Files and Directories 61
File Merge Algorithm 63
Determination of the Base Contributor 64
Recording of Merge Arrows 65
Locating Versions with Merge Hyperlinks 66
Directory Merge Algorithm 66
Merging Directories 66
Using ln and rmname in Diff Merge 67
Scenario: Merging All Changes Made on a Subbranch 67
Task Overview 67
Getting More Information 68
Scenario: Selective Merge from a Subbranch 68
Merging a Range of Versions 69
Task Overview 70
Getting More Information 71
Scenario: Removing the Contributions of Some Versions 71
Task Overview 71
Getting More Information 72
Recording Merges That Occur Outside ClearCase 72
Getting More Information 73
Sharing Control of a Branch with Developers at Other Sites 73
To Request Mastership of a Branch and Wait for the Transfer 75
To Check Out the Branch Before Mastership Is Transferred 75
Troubleshooting 76

Other Tasks 77
Adding Files and Directories to Source Control 77
To Add Elements to Source Control 77
Under the Hood: What Happens When You Add to Source Control 78
File Types and Element Types 79
Access Mode 79
Conversion of View-Private Files to Elements 80
Conversion of Nonshareable Derived Objects to Elements 80
Creation of Directory Elements 81
Auto-Make-Branch During Element Creation 81
Creation of Elements in Replicated VOBs 81
Element Object and Version References 82
Storage Pools 83
Group Membership Restriction 83
Importing Files 83
Moving, Removing, and Renaming Elements 83
Moving and Removing Elements 83
To Move an Element Within a VOB 84
To Move an Element to Another VOB 84
To Remove an Element Name from a Directory 84
Other Methods for Removing Elements 85
Renaming Elements 85
To Rename an Element 85
Accessing Elements Not Loaded into a Snapshot View 86
Listing All Elements in the VOB Namespace 86
Viewing the Contents of a Nonloaded Version 86
To Copy a Nonloaded Version with cleartool get 86
Registering a Snapshot View 87
To Register a Snapshot View 87
To Regenerate .ccase_svreg 87
Moving Views 88
Changing the Physical Location of a Snapshot View 88
To Find the Location of the View Storage Directory 88
Update After Moving 88
Moving a View Storage Directory 89
Regenerating a Snapshot View .view.dat File 89
To Regenerate the .view.dat File 89
Accessing Views and VOBs Across Platform Types 90
Creating Views Across Platforms of Different Types 90

Snapshot View Characteristics and Operating-System Type 90
Accessing Views Across Platforms of Different Types 91
Accessing UNIX Snapshot Views from Windows Hosts 91
Accessing Windows Snapshot Views from UNIX Hosts 91
Accessing UNIX Dynamic Views from Windows Hosts 91
Accessing Windows Dynamic Views from UNIX Hosts 91
Accessing VOBs Across Platforms of Different Types 92
Developing Software Across Platforms of Different Types 92

Working in a Snapshot View Without the Network 93

Setting Up a View for Your Hardware Configuration 94
Under the Hood: View Storage in Disconnected-Use Configurations 95
Preparing the View 96
Disconnecting the View 96
Working in the View 96
Hijacking a File 96
To Hijack a File 97
To Find Modified Files While Disconnected 97
Reconnecting to the Network 97
Using the Update Tool 97
Determining How to Handle Hijacked Files 97
To Find Hijacked Files 98
To Compare a Hijacked File to the Version in the VOB 98
To Check Out a Hijacked File 99
Merging the Latest Version to a Hijacked File 99
To Undo a Hijack 100
Under the Hood: Determining Whether a File Is Hijacked 100
Other Ways to Handle Hijacked Files 100
Updating the View 101

Index 103

Figures

Figure 1 A VOB Contains All Versions of an Element 3
Figure 2 Config Spec for Snapshot Views 5
Figure 3 Version Tree and Extended Namespace 9
Figure 4 Resolution of Reserved and Unreserved Checkouts 32
Figure 5 Selecting the Nonlatest Version of an Element 34
Figure 6 Merging the Latest Version and Your Checkout 44
Figure 7 Update Tool Window 51
Figure 8 Update Operation 53
Figure 9 Linear Progression of Versions 56
Figure 10 Version Tree of a File Element 57
Figure 11 The Initial Version on a Subbranch 57
Figure 12 Elements Have Common Branches 59
Figure 13 Version-Extended Paths 60
Figure 14 Versions Involved in a Typical Merge 62
Figure 15 ClearCase Merge Algorithm 63
Figure 16 Determination of the Base Contributor for a Merge 65
Figure 17 Merging All Changes from a Subbranch 68
Figure 18 Selective Merge from a Subbranch 69
Figure 19 Removing the Contributions of Some Versions 71
Figure 20 Creating an Element 79
Figure 21 View on a Laptop 94
Figure 22 View On a Removable Storage Device 94
Figure 23 Copy the View 95
Figure 24 Hijacked Files in the Update Window 98
Figure 25 Hijacked Version May Not Be the Latest Version 99

Rational ClearCase Concepts

1

Rational ClearCase provides a flexible set of tools that your organization uses to implement its development and change management policies. To use these tools, you need to understand the following concepts:

- ClearCase elements
- Views, versions, and VOBs
- Parallel development

If your project uses the base ClearCase-ClearQuest integration, you need to understand the following concepts:

- Activities
- ClearQuest user databases
- Associating versions and activities

Recommended Reading Paths

Read this chapter first. Then, if you want to start working immediately, use Help to learn as you go. Or, if you prefer a more structured approach, use the remainder of *Working in Base ClearCase* as a guide through the development cycle of your organization. The sections titled *Under the Hood* provide detailed information and suggest ways to become an advanced ClearCase user.

To Access Online Information

To start ClearCase Help, type the following command:

cleartool man contents

The help provides links to introductory material. To access the tutorials, type the following command:

cleartool man tutorials

For more information, see the discussion of online documentation in the Preface.

ClearCase Elements and Activities

Items under ClearCase source control (version control) are generally referred to as elements. An element can be a design model, C++ source file, Visual Studio project, or a DLL. Elements are typically the objects on which you do work.

If your project uses the base ClearCase-ClearQuest integration, the things that you do in performing your work and their status are captured in *activities*. Activities are stored in ClearQuest user databases. Activities represent work assigned to you, and allow your project manager to gauge your progress.

This section describes the following concepts in understanding elements and activities:

- Views
- Elements and versions
- ClearQuest user databases

ClearCase Views

To access files under ClearCase control, you set up and work in a *view*, which shows a directory tree of specific *versions* of source files.

Snapshot Views and Dynamic Views

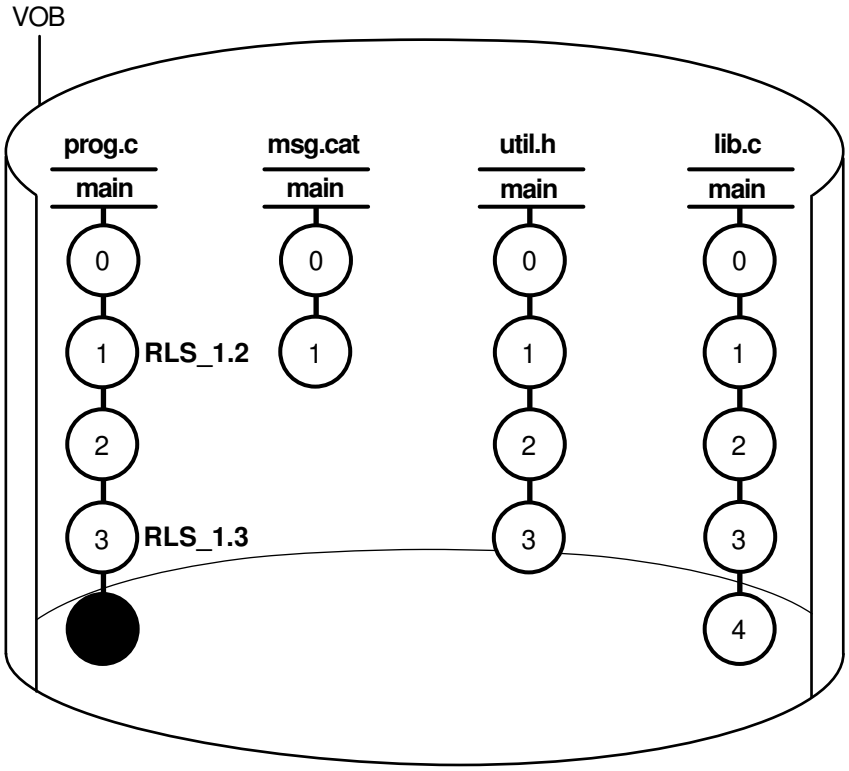
ClearCase includes two kinds of views:

- *Snapshot views*, which copy files from data repositories called *VOBs* (*versioned object bases*) to your computer.
- *Dynamic views*, which use the *multiversion file system* (MVFS) of ClearCase to provide immediate, transparent access to the data in VOBs. (Dynamic views may not be available on all platforms. For more information, see the Help. For information about starting Help, see *To Access Online Information* on page 1.)

Versions, Elements, and VOBs

Each time you revise and check in a file or directory from a view, ClearCase creates a new *version* of it. Files and directories under ClearCase control (and all of their constituent versions) are called *elements* and are stored in *VOBs*. Figure 1 illustrates a VOB that contains the file elements prog.c, util.h, msg.cat, and lib.c.

Figure 1 A VOB Contains All Versions of an Element



Depending on the size and complexity of your software development environment, ClearCase *elements* may be distributed across more than one VOB. For example, the elements used by the documentation group are stored in one VOB, while the elements contributing to software builds are stored in a different VOB.

Selecting Elements and Versions

A set of rules called a configuration specification, or *config spec*, determines which files are in a view.

Config Specs for Snapshot Views

Config specs for snapshot views contain two kinds of rules: *load rules* and *version-selection rules*. Figure 2 illustrates how the rules in a config spec determine which files are in a view. In its examples, this manual describes views created by a user named Pat, who is working on release 1.4 of a product called Cropcircle. For purposes of clarity, Pat's snapshot view names end in **_sv**.

Config Specs for Dynamic Views

Dynamic views use version-selection rules only (and ignore any load rules). A dynamic view selects all elements in all mounted VOBs, and then uses the version-selection rules to select a single version of each element. Instead of copying the version to your computer as a standard file, the view uses the *MVFS* (multiversion file system) to arrange the data selected in the VOB into a directory tree.

Criteria for Selecting Versions

The rules in the config spec constitute a powerful and flexible language for determining which versions are in your view. For example, version-selection rules can specify the following criteria:

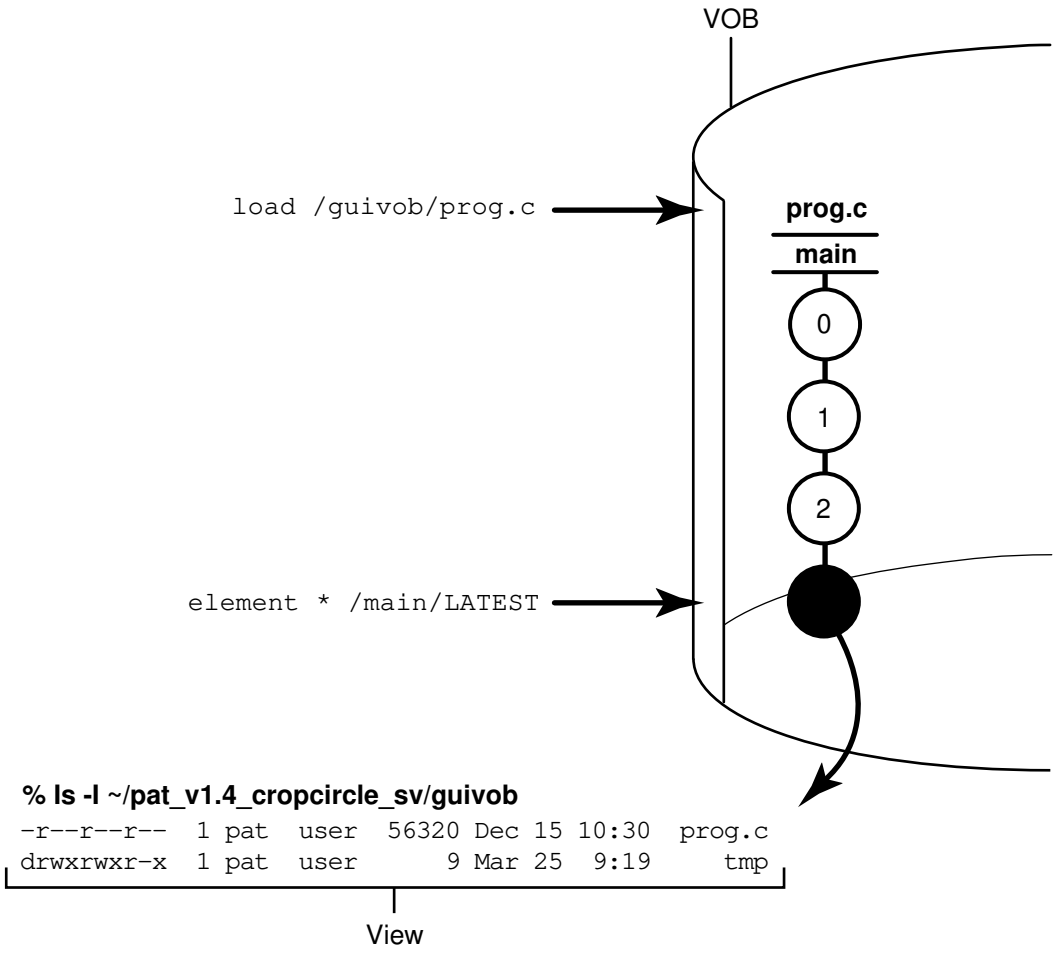
- The latest version.
- A version identified by a *label*.

  A label is a text annotation that you can attach to a specific version of an element. Usually, your project manager attaches a label to a set of versions that contributed to a specific build. A typical config spec rule uses version labels to select versions:

  ```
  element * BASELINE_1
  ```

  For example, if your project manager attaches version label **BASELINE_1** to a version of element prog.c, any view configured with this rule selects the labeled version (unless some rule earlier in the config spec matches another version of prog.c). For more information about labels, see *Managing Software Projects*.

Figure 2 Config Spec for Snapshot Views



- A version identified by a *time rule*, that is, a version created before or after a specific time.

The version-selection rules are prioritized. For example, the view can try to select a version identified by a label first, and if no such version exists, the view can select a version based on a time rule.

Version Labels in Version-Extended Paths

In addition to affecting the way the element appears in views, labeling a version of an element also provides a way to access the version with a version-extended path (see *The Version-Extended Path* on page 58). Labeling a version effectively adds a new hard

link to the version in the extended namespace. If you attach version label **R4.1A** to version **/main/rls4/12** of element lib.c, these paths are equivalent:

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

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

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

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

Learning the Config Spec Syntax

Usually only one or two members of your software team learn the syntax for these rules and create config specs for everyone on the project to use. For more information, see *Managing Software Projects* and the **config_spec** reference page in the *Command Reference*.

View-Private Objects

In addition to versions of source files, a view also contains file-system objects that are not under ClearCase source control, such as temporary files that you create while developing your source files and other objects. These non-ClearCase file system objects are called *view-private objects.*

The Base ClearCase-ClearQuest Integration

Your ClearCase administrator can integrate base ClearCase projects with a ClearQuest user database. You then associate versions of ClearCase elements on which you are working and change requests in a ClearQuest user database.

The Base ClearCase Schema and ClearQuest User Databases

ClearQuest stores data in schema repositories and user databases. A schema defines the types of records in the user database and other attributes of the user database. ClearQuest stores all schemas in a schema repository. The ClearQuest user database stores your change-request data.

The base ClearCase schema package provides additional information in your ClearQuest records to track associations with ClearCase versioned objects. To support associations between base ClearCase versions and ClearQuest record types, the

schema repository needs to have this schema package applied to selected record types. Then the target ClearQuest user database must be updated to add the new fields provided by the package.

Your ClearQuest user database may include different record types for different purposes. The record type used by the SAMPL ClearQuest user database supplied with the base ClearCase-ClearQuest integration is called a defect, but with the base ClearCase schema package installed, any change-request record type can be used.

ClearCase Triggers and Change Requests

The base ClearCase-ClearQuest integration consists of ClearCase triggers that fire when you check out an element, cancel a checkout, or check in an element. Your ClearCase administrator installs the integration triggers into each target VOB. The integration associates one or more change requests with one or more versions stored in one of the target VOBs.

- A single change request may be associated with more than one version. The set of versions that implement the requested change is called the *change set* for that request.
- A single version may be associated with more than one change request. These change requests are called the *request set* for that version.

Uses of the Base ClearCase-ClearQuest Integration

The base ClearCase-ClearQuest integration provides either of the following:

- A window interface for users of the graphic user interfaces (GUIs) of ClearCase, such as the File Browser of ClearCase (on UNIX workstations)
- A text-based user interface for users of the **cleartool** command-line interface

The base ClearCase-ClearQuest integration has triggers on checkin, checkout, and cancel checkout operations. Working on versions, you can do the following:

- Associate a version with one or more change requests when you check out or check in elements.
- List the request sets that are associated with a project over a period of time, list the change requests associated with a specific version, and see the related hyperlinks.

Using the ClearQuest user database, you can do the following:

- View the change set for a change request.
- See the elements that fix a specific problem.

ClearCase administrators can do the following:

- Install the related triggers in a VOB and set a policy for each VOB.
- Specify that you are prompted on checking out a version, checking in a version, or both.
- Specify that prompting occurs only for some VOBs, branch types, or element types. Associations of checked-in versions with change requests can be either optional or required.

A ClearCase administrator adds the base ClearCase schema package to a schema and applies the change to one or more change-request record types. The policy that the administrator sets for one or more VOBs specifies the conditions under which you are prompted to associate versions with change requests.

Parallel Development

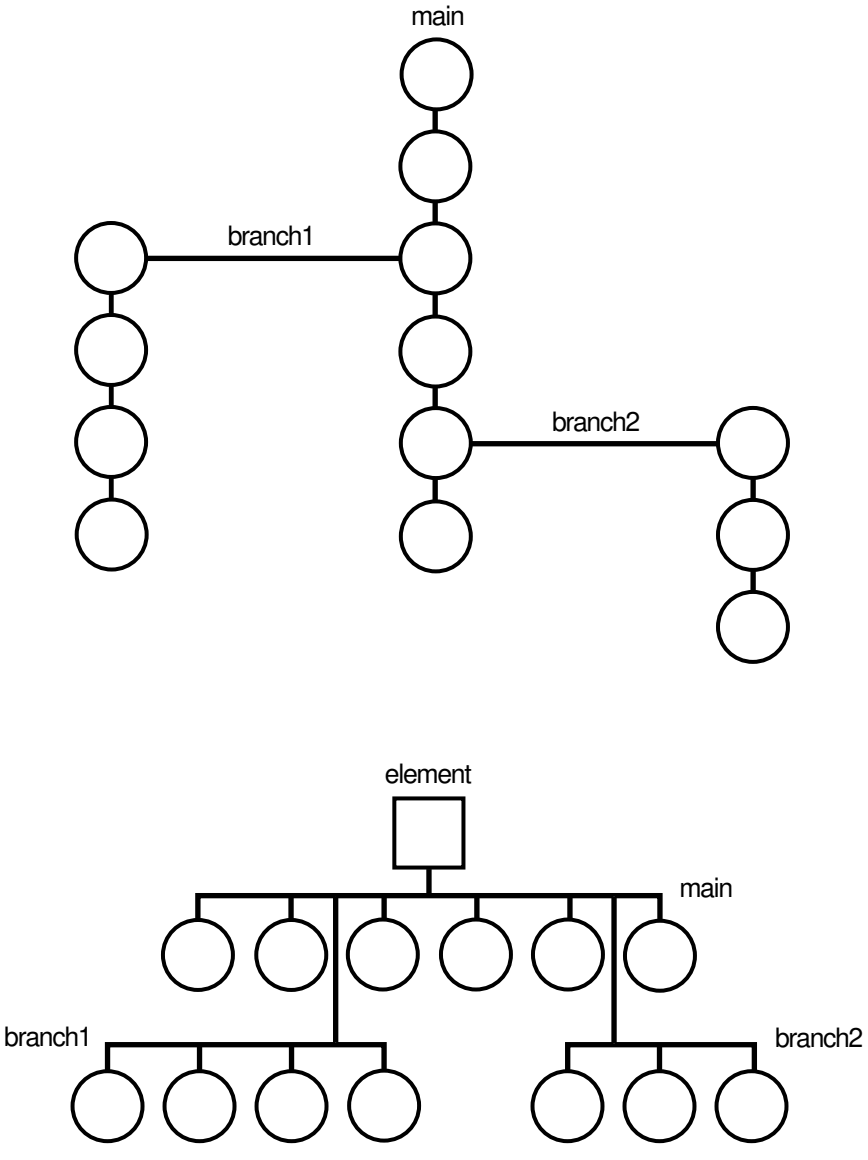
The combination of config spec rules, views, VOBs, and *branches* (described in Chapter 5, *Working On a Team*) provide the basis for *parallel development*, a strategy in which an organization can work on multiple versions of the same source file concurrently. For example, you are working on release 1.4 of a software product, and you want to experiment with the graphic user interface as a result of feedback from usability testing. You can create a view that isolates your modifications from the rest of the release 1.4 development project. Although you work with the same set of files used in the official builds, the versions of the files that you create from this view evolve separately from the versions used in the official builds. When you are satisfied with your usability modifications, you can use ClearCase tools to merge your work with the files used in the official release 1.4 build.

Extended Namespace for Elements, Branches, and Versions

A version tree for an element has the same form as a standard directory tree (see Figure 3), which compares components of the version tree to components of a directory tree in extended namespace.

As a component of the version tree, the element is the root of the directory tree in the extended namespace. The element itself appears to be a directory, which contains a single subdirectory, corresponding to the **main** branch. (It can also contain some version labels.)

Figure 3 Version Tree and Extended Namespace



A branch in the version tree appears as a subdirectory in the extended namespace. As a directory, each branch can contain files (individual versions and version labels), directories (subbranches), and links (version labels).

A version in the version tree is a leaf name of the directory tree in the extended namespace. Each version of an element is a leaf of the directory tree. For a file element,

the leaf contains text lines or binary data. For a directory element, the leaf contains a directory structure.

Accordingly, any location within the version tree of an element can be identified by a path in this extended namespace:

sort.c@@	*(specifies an element)*
sort.c@@/main	*(specifies a branch)*
sort.c@@/main/branch1	*(specifies a branch)*
sort.c@@/main/branch1/2	*(specifies a version)*
doctn/.@@/main/3	*(special case: extra component is required in top-level directory of VOB)*

Setting Up a Work Environment

2

Usually when you set up a work environment, you establish a separate view for each development project to which you contribute. Setting up a view involves the following tasks:

- Choosing snapshot view or dynamic view
- Choosing a location and name
- Adjusting your **umask**
- Using the **cleartool mkview** command
- Adding or modifying version-selection rules
- Adding or modifying load rules in a snapshot view

If your project uses the base ClearCase-ClearQuest integration, you access one RationalClearQuest user database. Accessing a ClearQuest user database involves the following tasks:

- Choosing a master schema
- Setting up a profile file (.ini)

Choosing a Snapshot View or a Dynamic View

Decide whether you want to work in a *snapshot view* or a *dynamic view*. As described in *ClearCase Views* on page 2, snapshot views load elements onto your computer; dynamic views use the *MVFS* to arrange VOB data into a directory tree. (Dynamic views may not be available on all platforms. For more information, see Help. To access Help, see *To Access Online Information* on page 1.)

Work in a snapshot view when any of these conditions is true:

- Your workstation does not support dynamic views.
- You want to work with source files under ClearCase control when you are disconnected from the network that hosts the VOBs.
- You want to simplify accessing a view from a workstation that is not a ClearCase host.
- Your development project does not use the *build auditing* and *build avoidance* features of ClearCase.

Work in a dynamic view when any of these conditions is true:

- Your development project uses build auditing and build avoidance.
- You want to access elements in VOBs without copying them to your workstation.
- You want the view to reflect changes made by other team members at all times (without requiring an *update* operation).

For more information, see the *Administrator's Guide* for Rational ClearCase.

Choosing a Location and Name

Before creating the view, you must choose its location. For a dynamic view, you must also choose a name. This section describes the following tasks:

- Choosing a location for a snapshot view
- Choosing a view name
- Choosing a location for a dynamic view storage directory

Snapshot View: Choosing a Directory

When creating a snapshot view, you must specify a directory into which ClearCase *loads* (copies) files and directories. When choosing a directory for the view, consider these constraints:

- The view root directory must be located on a disk with enough space for the files loaded into the view and any *view-private files* you add.
- Your organization may restrict where you can create a view. For example, you may be required to use a disk that is part of a data-backup scheme.
- If you want to access the view from other workstations, it must be located in a directory that is accessible to the other workstations; that is, choose a disk partition that is exported.

Under the Hood: A Snapshot View Storage Directory

Every snapshot view has a *view storage directory* in addition to the directory tree of source files that it loads from VOBs. ClearCase uses the snapshot view storage directory to keep track of such information as which files are loaded into your view and which versions are checked out to it. The view storage directory is for ClearCase administrative purposes only. Do not modify anything in it.

For every 1,000 elements loaded into the view, ClearCase uses about 400 KB of disk space for the view storage directory.

Locations for Snapshot View Storage Directories

Usually, your ClearCase administrator sets up a storage location, which is a directory on a ClearCase server host on UNIX or Windows. By default, ClearCase locates snapshot view storage directories there. If your ClearCase administrator sets up more than one storage location, ClearCase selects any one of these locations as the default when you create a view.

If your ClearCase administrator does not set up storage locations, by default, ClearCase software locates the view storage directory under the root directory of the snapshot view.

You can override these defaults. If your administrator sets up multiple storage locations, you can select one explicitly. You can place the view storage directory under the root directory of the snapshot view.

If you place the view storage directory under the root directory of the view, be aware of the following recommendations:

- Do not choose this configuration if you use the view when disconnected from the network. You can corrupt the data in the view storage directory if you disconnect it from the network while the *view_server* process of the view is running.
- Make sure that the view storage directory is accessible to any data backup schemes your organization institutes.

  Note: If you plan to work while disconnected from the network, your administrator must set up storage locations.

Choosing a View Name

Each view must have a descriptive name (called a *view tag*) that is unique within a network region. Choose a view name that helps you determine the owner and purpose of the view. Names like **myview** or **work** do not describe the owner or contents of the view; if you work with more than one view, such generic names can lead to confusion. Here are some suggested names:

pat_v1.4_cropcircle	Personal view for a user named Pat to develop source files for release 1.4 of the Cropcircle product
1.3_fix	Shared view for use in a particular bug-fixing task

The name of a view must be a simple name; that is, it must follow the format of a single file or directory name with no special characters or spaces.

Using the View Tag

The way you use the *view tag* is different for each type of view:

- ClearCase provides a default view tag for snapshot views based on the following convention:

  user-ID_leaf-of-view-path

  You do not refer to the view tag when performing most ClearCase operations. Instead you usually refer to the path of the view.

  The root directory of the snapshot view contains a file, .view.dat, which provides information that ClearCase uses to perform operations on the files in the view. When ClearCase finds the .view.dat file of the view, it can determine the view tag of the view. (If you delete the .view.dat file inadvertently, see *Regenerating a Snapshot View .view.dat File* on page 89.)

- For dynamic views, the view tag is the only name you use when performing most ClearCase operations. After setting (activating) a dynamic view, you use the view tag to refer to the root directory of the directory tree of the view. For more information, see *Accessing Files* on page 29 and the **pathnames_ccase** reference page in the *Command Reference*.

Dynamic View: Choosing a Location for the View Storage Directory

When creating a dynamic view, you must choose a location for its *view storage directory*. ClearCase uses this directory to keep track of which versions are checked out to your view and to store view-private objects. The view storage directory is for ClearCase administrative purposes only. Do not modify anything in it.

The size of the view storage directory depends on the following factors:

- Whether you use the **clearmake** *build auditing* and *build avoidance* features
- The size and number of view-private files

For more information, see the *Administrator's Guide* for Rational ClearCase and the **clearmake** reference page in the *Command Reference*.

Choosing Locations for Dynamic View Storage Directories

Consider the following restrictions when choosing a dynamic view storage directory location:

- The directory must be located on a ClearCase host or on a Network Attached Storage (NAS) device. View processes (specifically, *view_server* processes) run on the computer that physically stores the view storage directory or on the computer

that can access a location on a supported NAS device which contains the view storage directory. And only ClearCase hosts can run view processes.

- To maintain data integrity, the view storage directory must remain connected to the network. For example, do not locate the view storage directory on a removable storage device.
- If you locate the view storage directory on a laptop and then disconnect the laptop from the network, all of the following restrictions apply:
  - You cannot use the dynamic view.
  - Team members who try to start your view from their hosts will receive error messages from ClearCase.
  - Any **clearmake** process that attempts to wink in a derived object from your view will spend some amount of time trying to contact your view. If it cannot contact your view, it will not consider derived objects in your view as *winkin* candidates for 60 minutes. (You can change the amount of time by setting the CCASE_DNVW_RETRY environmental variable.) For more information, see the **clearmake** reference page.
- If you use the view on several hosts, make sure that the location can be accessed by all those hosts; that is, choose a disk partition that is exported.
- If your ClearCase administrator sets up storage locations (which are directories on ClearCase server hosts), you can locate your dynamic view storage directory in a storage location (with **mkview –stgloc**). However, for best performance, we recommend that you locate dynamic view storage directories on your local host.

We recommend that you make the view storage directory accessible to any data backup schemes your organization institutes.

Adjusting Your umask

Your **umask(1)** setting at the time you create a view affects how accessible your view is to others. For example:

- A umask of **002** is appropriate for a view that you share with other users in the same group. Members of your group can create and modify view-private data; those outside your group can read view-private data, but cannot modify it. To completely exclude nongroup members, set your umask to **007**.
- A umask of **022** produces a view in which only you can write data, but anyone can read data.

- A umask of **077** is appropriate for a completely private view. No other user can read or write view-private data.

Change your umask in the standard way. For example, enter this command from a shell:

umask 022

For more information, see a **umask(1)** reference page.

The CCASE_BLD_UMASK Environment Variable

You can also use the CCASE_BLD_UMASK environment variable to set the **umask(1)** value 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 is 22.

For more information about environment variables for ClearCase, see the **env_ccase** reference page in the *Command Reference*.

Creating the View with cleartool mkview

After gathering information on names and locations, open a shell and enter the **cleartool mkview** command as described in the following sections.

To Create a Snapshot View

Enter the following command:

cleartool mkview –snapshot *path-for-view*

Remember the path that you enter; ClearCase creates your directory tree of source files at this path.

For a complete list of **mkview** options, see the **mkview** reference page in the *Command Reference*.

For example, to create the **pat_v1.4_cropcircle_sv** view located under Pat's home directory, enter the following command:

cleartool mkview –snapshot ~/pat_v1.4_cropcircle_sv

```
Selected Server Storage Location "cropcircles_view_storage".
Created view.
Host-local path:
BREAD:/storage/cropcircles_view_storage/pat_v1.4_cropcircle_sv.vws
Created snapshot view directory "~/pat_v1.4_cropcircle_sv".
```

Under the Hood: .ccase_svreg

When you create a snapshot view, ClearCase creates or modifies the file .ccase_svreg in your home directory. Do not remove or relocate this file; some ClearCase operations require it.

If you inadvertently delete or corrupt this file, see *To Regenerate .ccase_svreg* on page 87.

To Create a Dynamic View

Enter the following command:

cleartool mkview –tag *dynamic-view-tag dynamic-view-storage-path*

For a complete list of **mkview** options, see the **mkview** reference page in the *Command Reference*.

For example, create **pat_v1.4_cropcircle** as a dynamic view with the following command:

cleartool mkview –tag pat_v1.4_cropcircle ~/pat_v1.4_cropcircle.vws

Under the Hood: The cleartool Command-Line Interface

cleartool is the command-line interface (CLI) to interact with ClearCase. It has a large set of subcommands, which create, modify, and manage the information in VOBs and views.

You can use **cleartool** in either single-command mode or interactive mode. To use a single **cleartool** subcommand from a shell, use this syntax:

cleartool *subcommand* [*options-and-args*]

When entering a series of subcommands, you may find it more convenient to type **cleartool** without any arguments. This places you at the interactive-mode prompt:

```
cleartool>
```

You can then issue any number of commands, ending with **quit** to return to the original shell. **cleartool** commands can be continued onto additional lines with backslash (\), as with UNIX shells.

Command options may appear in any order, but all options must precede any nonoption arguments (typically, names of files, versions, branches, and so on).

For more information, see the **cleartool** reference page in the *Command Reference*. For information about input and output for **cleartool** commands, see **pathnames_ccase** and **fmt_ccase** in the *Command Reference*.

Adding or Modifying Version-Selection Rules

ClearCase creates a set of default version-selection rules in the initial config spec of your view. However, development projects often require team members to add specific version-selection rules. This manual assumes that someone in your organization creates these rules, and you must either copy them into your config spec or add an inclusion rule so that your config spec includes them from a config spec available over the network. For information about creating version-selection rules, see *Managing Software Projects*.

You can use the following procedure whenever you need to add, remove, or otherwise modify the version-selection rules of your view.

To Copy or Include Version-Selection Rules

1. Do one of the following:
   - If you work in a dynamic view, type the following command:

     cleartool setview *view-tag*
   - If you work in a snapshot view, change to the root directory of the view.
2. Type the following command:

   cleartool edcs

   ClearCase opens the config spec of the view in your default text editor. For information about changing the default editor, see the **env_ccase** reference page in the *Command Reference*.
3. In your text editor, do any of the following:
   - Copy or insert the rules for your project into the config spec. The rules may be available in a text file accessible over the network, or even through e-mail. Verify with the author of the shared config spec whether you need to include any rules other than the ones you paste.
   - Type on its own line **include** *path-to-shared-config-spec*. Verify with the author of the shared config spec whether you need to include any rules other than the include rule.

   Note: Rather than create your own config spec with an **include** rule, you can use **cleartool setcs** to use some other config spec directly. For more information, see the **setcs** reference page in the *Command Reference*.
4. If you work in a snapshot view, create or modify *load rules*. (For more information, see *Snapshot View: Adding or Modifying Load Rules*.)

If you work in a dynamic view, do the following:

a Save the config spec and exit the text editor.

b In your shell, answer **Yes** at the prompt for setting the config spec.

Snapshot View: Adding or Modifying Load Rules

Load rules determine which elements are copied into the snapshot view. Any projectwide config spec that you include in the config spec for your view may already contain a set of default load rules.

You can modify those rules or add your own by doing the following:

- Listing the VOB namespace
- Adding or modifying load rules
- Excluding elements from loading

Listing the VOB Namespace

To create load rules, you must know the names of the elements in the VOB namespace. Because ClearCase loads directory elements recursively, you need to know only the names of parent directory elements.

VOB Namespace

In its general sense, a *namespace* is a set of unique names. The namespace of a file system usually consists of a hierarchical arrangement of files and directories. Each file and directory has a unique name.

In a VOB, a simple file name is not sufficient to select a single, unique object. For example, prog.c is ambiguous: does it refer to version 1 of prog.c or version 42 of prog.c?

A *VOB namespace* is the set of file and directory versions your config spec selects. For example, the view **pat_v1.4_cropcircle_sv** sees a VOB namespace of the files and directories that contribute to release 1.4 of a software product; the view **pat_v1.3_cropcircle_sv** sees a VOB namespace of the files and directories that contribute to release 1.3 of the same product. Because ClearCase tracks versions of directories, the VOB namespace varies depending on the versions of directories you select.

To List the VOB Namespace

1 Enter **cleartool lsvob** to see the list of VOBs at your site. For example:

cleartool lsvob –short

```
/guivob
/doc
```

2 To see the namespace in a VOB, enter the following command:

cleartool ls *snapshot_view_path/VOB-path* […]

To see further down a directory tree in the namespace, use **cleartool ls** recursively. For example:

cleartool ls pat_v1.4_cropcircle_sv/guivob

```
pat_v1.4_cropcircle_sv/guivob/batch@@main/1 [not loaded] Rule: /main/LATEST
pat_v1.4_cropcircle_sv/guivob/src@@main/1 [not loaded] Rule: /main/LATEST
```

cleartool ls pat_v1.4_cropcircle_sv/guivob/batch

```
pat_v1.4_cropcircle_sv/guivob/batch/prog.c@@main/1 [not loaded] Rule:
/main/LATEST
pat_v1.4_cropcircle_sv/guivob/batch/lib.c@@main/1 [not loaded] Rule:
/main/LATEST
```

The annotation **not loaded** indicates that there are no load rules specifying the element or that the version-selection rules do not select any version of the element.

Adding or Modifying Load Rules

You can add or modify load rules in any of the following ways:

- When editing the config spec. Any time you edit and modify the config spec for a snapshot view, ClearCase updates the entire view. This is appropriate when you first create a view, or when you modify the version-selection rules of a view, but it may be cumbersome if you only want to add a few elements to the view.
- By using **update –add_loadrules**. The **–add_loadrules** option of **cleartool update** adds load rules to the config spec for your view but updates only the portion of the view that is affected by the new load rules.

To Add or Modify Load Rules When Editing the Config Spec

1 Open the config spec of a view for editing by doing the following:

a Open a shell and change to a directory in the view.

b Enter the following command:

cleartool edcs

ClearCase opens the config spec of the view in your default text editor.

2 In your text editor, use the following syntax to create load rules:

load *vob-tag/element-path* […]

For example, the rule **load /guivob** loads all files and directories in the VOB named /**guivob**. The rule **load /guivob/batch** loads only the **batch** directory recursively.

3 Save the config spec and exit the text editor.

4 In your shell, answer **Yes** at the prompt for setting the config spec.

To Add Load Rules with update –add_loadrules

Enter the following command:

cleartool update –**add_loadrules** *element-path* […]

The variable *element-path* names an element in the VOB namespace at a path that is relative to a snapshot view. For example, the following command loads all elements in a VOB named **/guivob** into the view **pat_v1.4_cropcircle_sv**:

cleartool update –add_loadrules ~/pat_v1.4_cropcircle_sv/guivob/batch/lib.c

```
Loading "guivob/batch/lib.c" (100352 bytes).
Done loading "/guivob/batch/lib.c" (1 objects, copied 98 KB).
Log has been written to
"/net/triazol/export/home/lsweeney/pat_v1.4_cropcircle_sv/update.27-Ja
n-03.13:23:16.updt".
```

The load rule is added at the end of the config spec and the new config spec is set.

You can also use a relative path for the *element-path* argument. For example, these commands load all elements in **guivob**:

cd ~/pat_v1.4_cropcircle_sv
cleartool update –add_loadrules guivob

These commands load only the **batch** directory recursively:

cd ~/pat_v1.4_cropcircle_sv
cleartool update –add_loadrules guivob/batch

Excluding Elements from Loading

ClearCase loads all directory elements recursively. To exclude some elements from loading, you can use an element rule in the *config spec* that selects the initial version of an element. For all ClearCase elements, the initial version contains no data.

To Exclude Elements

1 Open the config spec of the view for editing:

   a Open a shell and change to a directory in the view.

   b Enter this command:

   cleartool edcs

2 In the text editor, create an element rule that specifies the initial version of the element you want to exclude by using the following syntax:

   element *vob-tag/element-path* **/main/0**

   For example, the element rule **element /guivob/interface /main/0** loads the empty version of the **interface** directory in **/guivob**, preventing any of its child elements from loading.

3 Save the config spec and exit the text editor.

4 In your shell, answer **Yes** at the prompt for setting the config spec.

Under the Hood: VOB Links

A VOB link makes a file element or directory element accessible from more than one location in the VOB namespace. There are two kinds of VOB links: *symbolic links*, which are available for file and directory elements, and *hard links*, which are available for file elements only. We recommend that you use VOB symbolic links instead of VOB hard links whenever possible.

You use the **cleartool ln** command to create VOB links. For more information, see the **ln** reference page in the *Command Reference*.

Symbolic Links and Hard Links in Dynamic Views

In *dynamic views* (which use the MVFS, or multiversion file system), VOB links behave similarly to symbolic links or hard links in a UNIX file system: symbolic links point to a file or directory element in a different location, and hard links are alternative names for a single file element.

You cannot check out a VOB symbolic link; you must check out the symbolic link target.

When you check out a hard-linked element from a given path, ClearCase considers other paths for the element as "checked out but removed." That is, to prevent you from modifying the element from multiple paths, ClearCase executes standard checkout behavior at only one path (the one from which you entered the **checkout** command), but does not create view-private files at other paths. For information about standard checkout behavior, see the **checkout** reference page in the *Command Reference*.

Symbolic Links in Snapshot Views

Snapshot views created from a UNIX host maintain standard symbolic link behavior. In the context of loading a snapshot view, links are treated as VOB links (those that point to objects inside the VOB) and non-VOB links (those that point outside the VOB).

Hard VOB links are followed; symbolic links are copy-created. If a VOB link cannot be resolved, an error results. Non-VOB links are resolved, if possible, but it is not an error if they cannot be resolved.

> **Note**: When you create a snapshot view from a UNIX host, ClearCase assumes that the file system that contains the view supports symbolic links. If your file system does not support symbolic links, ClearCase reports errors if it encounters VOB links during the update operation.

If a *load rule* selects a symbolic link, ClearCase copies the link as well as the link target into the snapshot view (regardless of whether a load rule selects the link target). As with dynamic views, you cannot check out a symbolic link; you must check out the symbolic link target.

Hard Links in Snapshot Views

Instead of creating hard links in a snapshot view, each time a load rule selects a hard link, ClearCase loads the element into the view as a standard file.

Caution: Losing Data Because of VOB Hard Links

If you load multiple instances of a hard-linked element into a snapshot view, you must be careful to check out, modify, and check in only one instance of the file. When you check in a hard-linked file, ClearCase updates all other instances in your view, which could result in loss of data if you modified multiple copies of the same file. (Note that, when updating instances of files because of a checkin, ClearCase renames any *hijacked* file to *filename*.keep before updating it.)

For example, the following sequence of events will lead to lost data:

1. You check out the hard-linked file src/util.h.
2. ClearCase removes the read-only permission from util.h in the **src** directory only (which is the location from which you issued the **checkout** command).
3. You modify src/util.h but do not check it in.
4. Later, you lose track of which file you checked out. You then remove the read-only permission and modify util.h in the **temp** directory.
5. You check in temp/util.h. Even though you checked out and modified src/util.h, ClearCase does not prevent you from checking in temp/util.h; with a VOB hard link, temp/util.h is just another name for src/util.h.
6. Any changes you made to src/util.h are lost upon checkin because ClearCase updates all copies of duplicated files when you check in an element. Note that

ClearCase does not consider any copy of util.h to be hijacked (even if you change permissions), because you checked out the element in the VOB.

Setting Up ClearQuest User Database Defaults

If your project uses the base ClearCase-ClearQuest integration (see *The Base ClearCase-ClearQuest Integration* on page 6), the tasks that you perform can be associated with records in a ClearQuest user database. For each ClearQuest user database that you access, you can set up defaults for the following purposes:

- Controlling the appearance of the ClearQuest client display
- Receiving e-mail notification of ClearQuest user database changes
- Determining the print format of result sets
- Specifying start-up and operational actions

For information about accessing a ClearQuest user database, see *Logging On to a ClearQuest User Database* on page 35.

To Control the Display Appearance

You can control the appearance of the Result set and whether the SQL editor displays. To change the Result set appearance, do the following:

1. In the ClearQuest client, click **Edit > Grid Properties**. The Display Settings dialog box appears.
2. Select or clear an option. The **Preview** area shows the effect of the option setting.

   To save the setting in the ClearQuest user database profile, select **Save settings to profile**.
3. Click **OK**.

To control the display of the SQL editor, click **View** and select or clear **View SQL pane**.

The settings take effect immediately.

To Enable EMail Notification

Your project manager can define hooks in the ClearQuest user database to enable mail messages to be sent to users when certain state transitions occur in ClearQuest records. To provide the information for mail to reach you, do the following:

1. In the ClearQuest client, click **View > E-mail Options**. The Choose Email Provider page opens.

2 Select **Enable Email notification** to have mail messages sent to you and to have others notified of changes that you make.

3 In **Email Provider**, select the transport to use. Then click **Next**. The Configure Mail Server page opens.

4 Do one of the following:

- If you selected the SMTP transport, in **Outgoing SMTP Server**, enter the name of the mail server that your ClearQuest administrator tells you to use.

  In **Your Email Address**, enter the user and domain name of your mailbox (this address appears in the From field of messages triggered by your actions on a record and sent to others).

- If you selected the MAPI transport, in **Choose MAPI Profile**, select one from the list.

5 Click **Finish**. The Configure Mail Server page closes.

6 In the ClearQuest client, click **View > Change User Profile**. The User Profile dialog box opens.

7 In **Email**, enter the user and domain name of your mailbox.

  This e-mail address is used in the Send field of messages directed to you. It differs from the similar field that you enter if you use the SMTP transport. This field is useful to specify the address of a group of users who share a common interest.

8 Click **OK**.

Notification is immediately enabled.

To Set Defaults for Printing a Result Set

To support printing, you can associate a report format with a record type, tailor the appearance of the Result set, and determine the location of date and paging information at the top and bottom of report pages.

Click **File > Print Preview** to see the effects of the settings that are described in this section.

To Associate a Format and Record Type

To associate a report format for printing with the record type of the ClearQuest user database that you have opened, follow these steps:

1 In the ClearQuest client, click **File > Print Record Setup**. The Print Record Setup dialog box appears with the name in **Record Type** inactive, and, if any format association is current, the path to that format.

2 Click **Associate Format**. The Select Report Format dialog box appears.

If you have a current association, click **Undo Association** and select another format.

3 Navigate to and click the report format entry and click **OK**.

To Change Result Set Print Appearance

To tailor the appearance of the result set in the print output, follow these steps:

1 In the ClearQuest client, click **File > Page Setup**. The Page Setup dialog box appears.

2 Select and clear the desired settings

To save the setting in the ClearQuest user database profile, select **Save settings to profile**.

3 Click **OK**.

To Set Headers and Footers

To set paging information in the print output, follow these steps:

1 In the ClearQuest client, click **File > Headers/Footers**. The Header/Footer dialog box appears.

2 Use the **Header** tab to define the alignment of text appearing at the top of each page and the **Footer** tab to specify the contents at the bottom of each page.

To save the setting in the ClearQuest user database profile, select **Save settings to profile**.

3 Click **OK**.

To Set Start-Up and Operational Defaults

When working in a ClearQuest user database, you can run queries when the database starts or can use optionally use the query wizard.

To Run Queries at Startup

In the ClearQuest client Workspace pane, right-click on a query to run and click **Run at Startup**. Note that you may run multiple queries at startup, but startup time increases with each query that runs.

To Use the Query Wizard

When you click **Query > New Query**, the Choose Record Type dialog box appears. You select a record type and click **OK**. If **Use Query Wizard** on the **Query** menu is clear, the **Display editor** tab in the Query builder pane becomes active so that you can create a query. If **Use Query Wizard** on the Query menu is selected, the ClearQuest Query Wizard starts.

To use the query wizard, click **Query** and select **Use Query Wizard**. The next time you create a new query, the Query Wizard starts.

Working in a View 3

This chapter guides you through the everyday tasks of managing source files from Rational ClearCase:

- Accessing files
- Checking out elements
- Working with checkouts
- Canceling checkouts
- Checking in elements

Accessing Files

Because snapshot views and dynamic views use different methods for creating directory trees, the procedure for accessing source files differs for the two view types.

In a Snapshot View

Recall that when you create the view, ClearCase copies one version of each element specified by a *load rule* into your view. To access the files loaded into a snapshot view, open a shell and change to the root directory of the view.

For example, this command creates the snapshot view:

cleartool mkview –tag pat_v1.4_cropcircle_sv –snapshot ~/pat_v1.4_cropcircle_sv

The files in the view are located in the ~/pat_v1.4_cropcircle_sv directory.

Accessing Someone Else's Snapshot View

You can access someone else's snapshot view as you would access any other directory on another workstation. Assuming that you can access the other workstation and that the owner of the directory has set up the proper permissions, use the **cd** command to access the view.

In a Dynamic View

Accessing source files from a dynamic view entails two procedures:

- Setting a view
- Mounting VOBs

To Set a Dynamic View

Type this command:

cleartool setview *view-tag*

For more information about setting a view, see the **setview** reference page in the *Command Reference*.

To Mount VOBs

Type this command:

cleartool mount *VOB-tag*

Usually, ClearCase mounts VOBs that were created with a public VOB tag when you start or reboot your workstation. If public VOBs do not mount, type **cleartool mount –all** to mount them.

VOBs remain mounted until you reboot your workstation or unmount them with the **cleartool umount** command. For more information about mounting VOBs, see the **mount** reference page in the *Command Reference*.

Accessing Someone Else's Dynamic View

Team members can access any dynamic view by starting it on their computers. If you are unable to start or set a dynamic view that is on another host, check with your administrator to make sure that you can access the view storage directory for the view. For more information, see the *Administrator's Guide* for Rational ClearCase.

Checking Out Elements

To modify files and directories under ClearCase control, you must check them out. (Placing files and directories under source control is a separate procedure; see *Adding Files and Directories to Source Control* on page 77.) If you work in an environment with the base ClearCase-ClearQuest integration, you may have to perform additional steps (see *Checking Out Elements in a VOB Enabled for ClearQuest* on page 34).

To Check Out an Element

1 In a view, enter this command:

cleartool checkout –query *list-of-elements*

ClearCase prompts you to enter a comment.

2 Describe the changes you plan to make.

3 To finish entering comments, press RETURN, and type a period or press CTRL+D on a blank line.

You can cancel the checkout operation by entering a standard interrupt signal such as CTRL+C before typing a period or pressing CTRL+D.

cleartool checkout includes several options. These are most commonly used:

–query
Detects potential problems in the checkout process caused by inappropriate config specs or out-of-date snapshot views and prompts for action.

–nc
Prevents ClearCase from prompting for a comment.

–cq
Prompts for and applies a comment to all elements in the list.

–unreserved
Makes the checkouts for the listed elements *unreserved*. For more information, see *Reserved and Unreserved Checkouts* on page 31.

For a complete description of all checkout options, see the **checkout** reference page in the *Command Reference*.

Reserved and Unreserved Checkouts

In some version-control systems, only one user at a time can reserve the right to create a new version. In other systems, many users can compete to create the same new version. ClearCase supports both models by allowing two kinds of checkouts: reserved and unreserved.

The view with a reserved checkout has the exclusive right to check in a new version for a given development project. Many views can have unreserved checkouts. An unreserved checkout does not guarantee the right to create the successor version. If several views have unreserved checkouts, the first view to check in the element creates the successor; developers working in other views must merge the checked-in changes into their own work before they can check in. The development policy of your organization may determine whether to check out reserved or unreserved.

Figure 4 illustrates checked-out versions created by reserved and unreserved checkouts, and the effects of subsequent checkins.

Another kind of checkout is an unreserved, nonmastered checkout, which can be used only in a replicated VOB (created with Rational ClearCase MultiSite). For more information about this kind of checkout, see *Sharing Control of a Branch with Developers at Other Sites* on page 73.

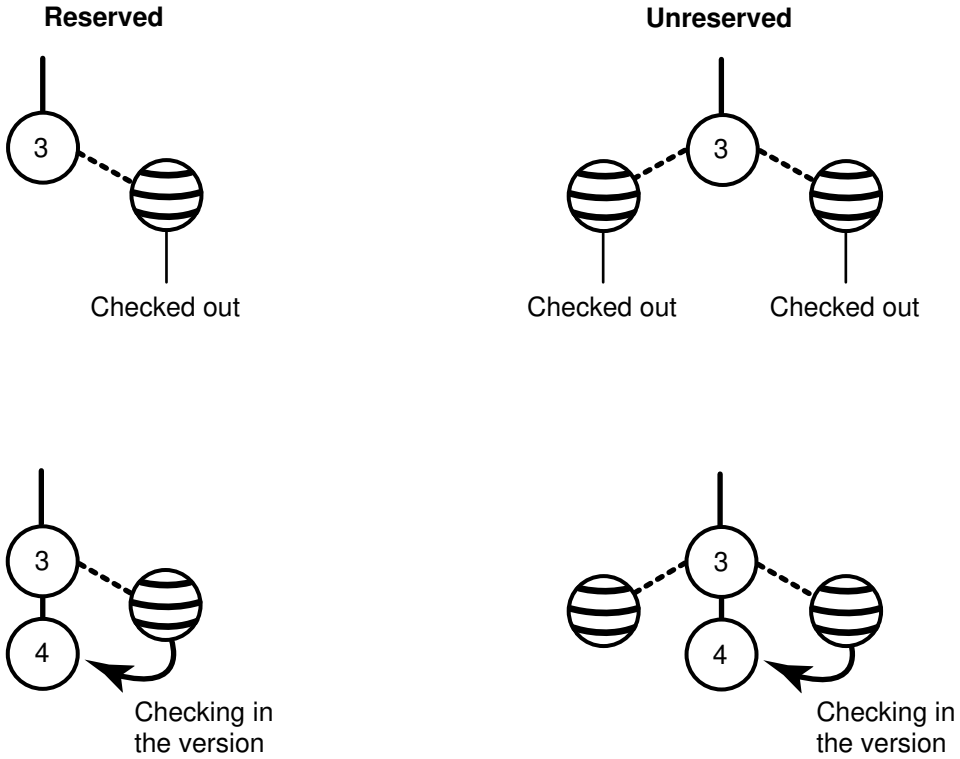
To Change the Status of a Checked-Out Version

In the view, enter the **reserve** or **unreserve** command, as follows:

- **cleartool reserve** *element-name*
- **cleartool unreserve** *element-name*

For information about changing the status for checkouts in other views, and for more information about these commands, see the **reserve** or **unreserve** reference pages in the *Command Reference*.

Figure 4 Resolution of Reserved and Unreserved Checkouts



Under the Hood: What Happens When You Check Out a File or Directory

Because a snapshot view contains copies of files and directories, and a dynamic view provides access to data in VOBs, ClearCase follows different procedures for checking out from the different view types.

Checking Out From a Snapshot View

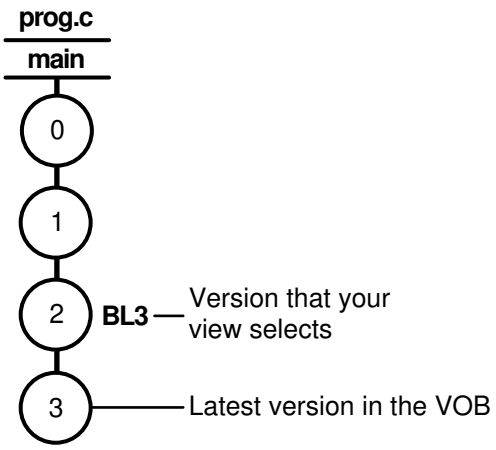
When you check out a file or directory from a snapshot view, the request is handled as follows:

1. It gathers the following information:
   - The version currently loaded in the view
   - The version selected by the config spec
   - The latest version in the VOB
2. If the version in your view is not the latest in the VOB, ClearCase notifies you. If you use the **–query** option when checking out a file, ClearCase asks you to specify which version to check out. If you use the **–query** option when checking out a directory, ClearCase notifies you, but requires you to check out the version of the directory currently loaded in your view.

   The version in your view will not be the latest in the VOB if either of these conditions exist:
   - Someone else has checked in a new version since you last updated your view.
   - The config spec of your view selects versions based on a label or a time rule, and the latest version in the VOB falls outside those parameters (Figure 5).
3. If you check out a version other than the one currently loaded in your view, ClearCase loads the checked-out version into your view.
4. ClearCase notifies the VOB which version of the element you checked out.
5. For files, ClearCase makes them writable. For directories, it allows you to use the **mkelem** command to add new elements to source control.

For information about checking out VOB links in a snapshot view, see *Under the Hood: VOB Links* on page 22.

Figure 5 Selecting the Nonlatest Version of an Element



Checking Out From a Dynamic View

When you check out a file from a dynamic view, ClearCase handles the request as follows:

1 If the version-selection rules for your view do not select the latest version in the VOB and you use the **–query** option with the **checkout** command, ClearCase prompts you to choose a version to check out.

Your view may not select the latest version in the VOB if, for example, your config spec selects versions based on labels or time rules (Figure 5).

If you do not use the **–query** option, ClearCase checks out the latest version without notifying you. Use the **–ver** option of the **checkout** command to check out the version that your view selects, even if it is not the latest in the VOB.

For information about checking in a version that is not the latest on the branch, see *Merging the Latest Version with Your Changes* on page 43.

2 ClearCase notifies the VOB which version of the element you checked out.

3 For files, ClearCase creates in the view an editable view-private file, which is a copy of the checked-out version. For directories, it allows you to use the **mkelem** command to add new elements to source control.

Checking Out Elements in a VOB Enabled for ClearQuest

If the VOB in which you access versions is set up for the base ClearCase-ClearQuest integration, you may have to associate the version on which you are working with a

ClearQuest record. For more information, see *The Base ClearCase-ClearQuest Integration* on page 6.

Logging On to a ClearQuest User Database

The first time that you attempt to check out an element from or check in an element to a VOB enabled for the base ClearCase-ClearQuest integration, you are prompted to log in to ClearQuest. Specify a ClearQuest user ID, password, and database name. ClearQuest keeps its own database of user IDs and passwords. Make sure that your ClearQuest administrator has added a user ID and password to the ClearQuest user database for you.

After you log in to ClearQuest, you can use the integration to complete your checkout and checkin operations (see *Using the Modified Graphic User Interface*). The integration stores the user ID and an encoded version of the password in a file named .cqparams, which it creates in platform-specific areas. On UNIX workstations, the file is stored in the home directory.

Thereafter, the integration uses the user ID and password in .cqparams instead of prompting you for them. If you change your password or connect to a different ClearQuest user database, the next time you check out or check in a version from that VOB, the integration displays an error message and prompts you to reenter the user ID and password. The integration updates the .cqparams file with the new information.

The Base ClearCase-ClearQuest Integration Interfaces

If you are logged in to a ClearQuest user database (see *Logging On to a ClearQuest User Database* on page 35) and check out or check in an element in a VOB enabled for ClearQuest, you see a modified interface, one for a graphic user interface or one for a command line interface. If you use File Browser, the Association dialog window opens (see *Using the Modified Graphic User Interface* on page 35). If you use a **cleartool** command line interface (CLI), you see a numbered menu (see *Using the Modified Command Line Interface* on page 36).

Using the Modified Graphic User Interface

For the modified graphic user interface, the integration is based on a Perl trigger. The Association dialog window displays the result set of a standard ClearQuest query, usually showing all open change-request IDs currently assigned to you. If local policies allow, you can select alternative site-specific or ClearQuest-supplied queries to run.

Select an entry in the result set and click **Associate** to create an association between the versions you are checking out and the change request. If multiple selections are allowed on your project, select multiple entries. With at least one association, click **OK**

to continue the checkout. To close the dialog box without making a change, click **Cancel**. For more information, click **Help** in the window.

Using the Modified Command Line Interface

For the modified **cleartool** command line interface, there are either text menus with numbered options or a **clearprompt** window showing the same menus. The options in the menus are shown in Table 1.

Table 1 Base ClearCase-ClearQuest Integration Options (Part 1 of 2)

Option	Description
OK - commit associations	Make the requested associations and exit.
CANCEL	Exit without making any changes and cancel the related ClearCase operation.
HELP	Display this text.
REVIEW Associations	Shows currently selected associations and allows you to delete one or more items from the list.
QUERY – Select from ClearQuest Query	Displays the contents of a query that your ClearCase administrator defines to show defects that are appropriate. Depending on your local policy, you select one or multiple items.
QUERYNAME	Shows the current query in Query. If this option appears, you see a list of alternative queries either defined by the configuration file (LOCAL) or in the ClearQuest user database itself. Your site administrator determines which queries appear.
TYPEIN – Enter Associations from Keyboard	Allows you to enter one or more change-request IDs directly.
DATABASE	Shows the current ClearQuest user database name; allows you to change to a different database.

Table 1 Base ClearCase-ClearQuest Integration Options (Part 2 of 2)

Option	Description
RECORD TYPE	Shows the current record type with which you are working; for example, a defect. Allows you to change to a different record type if multiple entities are supported by the current ClearQuest user database.
PATHNAME	Shows the full path for the version that you are checking in or checking out. Select this item to see more information.

To run an option:

- In the text menu, type the number of the entry and press RETURN.
- In the **clearprompt** window, select an option and click **OK**.

All menus accept a single choice, to which you can enter a number. TYPEIN allows multiple change-request IDs, separated by a space or a comma. The IDs can be full names (for example, SAMPL00056789) or numbers (for example, 56789).

To dismiss the menu without making a choice, simply press RETURN.

Using the Menu to Associate a Checkout with a ClearQuest Entity

If you use the command line interface and are required to associate versions with change requests, use the options from Table 1 to enter change-request IDs as follows:

- Use the QUERY option to see a list of all open change-request IDs currently assigned to you and select one or more items from the list to make associations.
- Use the TYPEIN option to enter one or more change-request IDs with which to associate the version that you are checking out.
- Use the REVIEW option to list and possibly delete current associations.
- If the association is optional and you do not want to specify a change request, enter the OK option and click **Yes** (**clearprompt**) or press RETURN (text menu).
- To cancel the checkout operation, use the CANCEL option or click **Abort**.
- To display help text, use the HELP option.

After you specify your options, use the **OK** option to create or update the associations that you specified and complete the checkout operation.

Working with Checkouts

After you check out a version, you do not need to interact with ClearCase until you are ready to check in. However, some ClearCase tools can help you with the following tasks:

- Viewing the history of an element
- Comparing versions of elements
- Tracking checked-out versions

Some **cleartool** commands include a –**graphical** option, which starts a tool for completing the task. This chapter presents the **–graphical** option whenever it is available.

Viewing the History of an Element

The History Browser displays the history of an element modifications, including version-creation comments (entered when someone checks out or checks in an element).

To View Element History

In a view, enter this command:

cleartool lshistory –graphical *path*

You can use this command from a snapshot view whether or not the element specified by *path* is loaded into the view.

Comparing Versions of Elements

As you modify source files, you may want to compare versions to answer such questions as these:

- What changes have I made in my checked-out version?
- How does my checked-out version differ from a particular historical version or from the version being used by one of my team members?

To Compare with a Predecessor

In a view, enter this command:

cleartool diff –graphical –predecessor *path*

To Compare with a Version Other Than the Predecessor

1. In a shell, enter this command:

   cleartool lsvtree –graphical *path*

2. In the Version Tree Browser, select a version.
3. Click **Version > Diff > Selected vs. Other**. The Enter other versions dialog box appears.
4. Select other versions and click **OK**.

To use the command line:

1. Use **cleartool lsvtree** to list the versions of an element.
2. Use the **cleartool diff** command with *version-extended naming*. For example, to compare the current version of prog.c with version 4**:**

   cleartool diff prog.c prog.c@@/main/4

You can use the **lsvtree** and **diff** commands from a snapshot view whether or not the element specified by *path* is loaded into the view. For more information, see *The Version-Extended Path* on page 58.

To Compare with a Version in a Dynamic View

Note: To use this procedure, your workstation must support dynamic views.

1. Use **cleartool startview** to start a dynamic view. For example, to compare a version in your view with a version in a dynamic view named **joe_v1.4_cropcircle**, enter the following command:

   cleartool startview joe_v1.4_cropcircle

2. Use **cleartool diff** (or any other **diff** command) with view-extended naming. For example, to compare /guivob/prog.c in your view with /guivob/prog.c in **joe_v1.4_cropcircle**, enter the following command:

   cleartool diff –graphical /guivob/prog.c /view/joe_v1.4_cropcircle/guivob/prog.c

Sometimes, the same element appears at *different* paths in different views. ClearCase can track directory-level changes, from simple renaming operations to wholesale reorganizations. In such situations, a team member may direct your attention to a particular element by using a path that is not valid in your view.

To determine the path in your own view, pass the path in the different view to a **describe –cview** command. For example:

cleartool describe –cview /view/joe_v1.4_cropcircle/project/include/lib.c

```
version "/guivob/lib.c@@/main/1"
  created 19-May-02.14:46:00 by rick (rick.devt@saturn)
  .
  .
```

You can then compare your version of the element with your team member's version as follows:

**cleartool diff –graphical /guivob/lib.c@@/main/1 **
/view/joe_v1.4_cropcircle/project/include/lib.c

Tracking Checked-Out Versions

Depending on how you work, you may forget exactly how many and which files are checked out. To list all the files and directories you currently have checked out to your view, access the view and use the **lscheckout** command with the following options:

cleartool lscheckout –cview –me –avobs

For more information, see the **lscheckout** reference page in the *Command Reference* or type **cleartool man lscheckout** in a shell.

Prototype Builds

Typically, when you are developing source files for a project, you want to perform prototype builds to test your modifications. If your organization uses **clearmake**, you can use this ClearCase build tool for your prototype builds; however, the *build auditing* and *build avoidance* features are available only from dynamic views.

For more information, see *Building Software* and the **clearmake** reference page in the *Command Reference*.

Canceling Checkouts

If you check out a file but do not want to check in your changes or want to start with a fresh copy, you can cancel the checkout as follows:

1. In the view from which you checked out a file, enter this command:

   cleartool uncheckout *path*

   ClearCase prompts you to save your modifications in a view-private file with a .keep extension.

2. To save the modifications in a view-private file, press RETURN. Otherwise, type **no**.

To avoid being prompted about saving modifications, use one of the following options with the **uncheckout** command:

–keep
Saves modifications

–rm
Does not save modifications. Any changes you made to the checked-out version are lost.

Under the Hood: Canceling Checkouts

When you cancel the checkout of a file element, ClearCase handles the request as follows:

1. It prompts you to rename the file in your view to *filename*.keep.
2. It notifies the VOB that you no longer have the version checked out in your view.
3. In a snapshot view, it copies from the VOB the version that was in your view when you performed the checkout operation.

   In a dynamic view, it uses the version-selection rules of the config spec to select a version.

If you work in an environment with the base ClearCase-ClearQuest integration, any associations with ClearQuest change requests you may have made at checkout (see *Checking Out Elements in a VOB Enabled for ClearQuest* on page 34) are canceled if you cancel the checkout.

Canceling Directory Checkouts

When you cancel a directory checkout, ClearCase notifies the VOB that you no longer have the version of the directory checked out to your view. ClearCase does not prompt you to rename a canceled directory checkout to *directory-name*.keep.

If you cancel a directory checkout after changing its contents, any changes you made with **cleartool rmname**, **mv**, and **ln** are lost. Any new elements you created (with **mkelem** or **mkdir**) become orphaned. ClearCase moves orphaned elements (and any data that exists in the view at the path of the new element) to the lost+found directory in the VOB under names of this form:

element-name.UUID

In such cases, **uncheckout** displays this message:

```
cleartool: Warning: Object "prog.c" no longer referenced.
cleartool: Warning: Moving object to vob lost+found directory as
"prog.c.5f6815a0a2ce11cca54708006906af65".
```

In a snapshot view, ClearCase does not remove *view-private objects* or start the update operation for the directory in the view. To return the directory in your view to its state before you checked it out, you must start the Update Tool. For information about starting the Update Tool, see *To Start the Update Tool* on page 50.

In a dynamic view, ClearCase does not remove view-private objects, but it does revert the view to its previous state.

To Move and Delete Orphaned Elements

To move an element from the lost+found directory to another directory within the VOB, use the **cleartool mv** command. To move an element from the lost+found directory to another VOB, use the **relocate** command. For more information about moving elements to another VOB, see the **relocate** reference page in the *Command Reference*.

To permanently delete an element in the lost+found directory, take note of the name of the orphaned element and use this command:

cleartool rmelem *VOB-path*/lost+found/*orphaned-element-name*

For example, from a dynamic view:

cleartool rmelem /guivob/lost+found/prog.c.5f6815a0a2ce11cca54708006906af65

From a snapshot view:

cd ~/pat_v1.4_cropcircle_sv
cleartool rmelem guivob/lost+found/prog.c.5f6815a0a2ce11cca54708006906af65

> **Note**: In a snapshot view, ClearCase treats the lost+found directory, which is located immediately below the root directory of a VOB, as any other directory. To load the directory in your view, you must use a load rule that specifies either the parent directory of the element or the directory itself. However, as with any other directory in a snapshot view, you do not need to load the lost+found directory to issue ClearCase commands for elements in the directory.

Checking In Files

Until you check in a file, ClearCase has no record of the work in your view. Checking in a file or directory element creates a new version in the VOB, which becomes a permanent part of the history of the element. We recommend that you check in a file or directory any time you want a record of its current state.

Ideally, the development strategy of your organization isolates your checked-in work from official builds and requires you to merge your work to official project versions at specified intervals.

To Check In Files

1. In a view, enter the following command:

   cleartool checkin *list-of-elements*

   ClearCase prompts you to append your checkout comments.

2. Type any additional comments, press RETURN, and type a period or press CTRL+D on a blank line.

   You may cancel the checkin operation by entering a standard interrupt signal such as CTRL+C before typing a period or pressing CTRL+D.

cleartool checkin includes several options. Here is a description of the most commonly used ones:

–nc
: Prevents ClearCase from prompting for a comment.

–cq
: Prompts for and appends a single additional comment to all elements in the list.

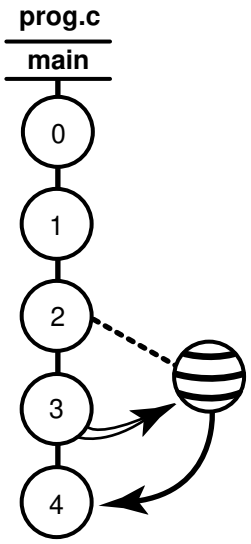
For a complete description of all checkout options, see the **checkin** reference page in the *Command Reference*.

Merging the Latest Version with Your Changes

If the version you checked out is not the latest version in the VOB and you try to check in your modifications, ClearCase requires you to merge the changes in the latest version into the version checked out in your view (Figure 6).

In Figure 6, version 2 of prog.c is the one that you checked out. Before you check in your modifications, someone else checks in version 3 of prog.c. When you check in your modifications, ClearCase tells you that the version you checked out is not latest on the branch. (For more information about situations in which you may have to merge before checking in, see *Under the Hood: What Happens When You Check Out a File or Directory* on page 33.) Note that the reserve status of the checkout is not relevant to whether your modifications can be checked in.

Figure 6 Merging the Latest Version and Your Checkout



You need to merge the latest version in the VOB (prog.c@@/main/LATEST) to the version in your view before you can check in your modifications. This merge creates a version that reconciles modifications made in the latest version with your modifications. Then, when you check in the merge results, the system sees the merge arrow from version 3 to your checked-out version containing the merge results. The checkin creates a version 3 successor, version 4 of prog.c.

To Merge the Latest Version

To merge the latest version in the VOB to the version in your view, enter the following command:

cleartool merge –graphical –to *file-or-directory-in-your-view* \
*file-or-directory-name***@@/main/LATEST**

> **Note**: @@/main/LATEST is a *version-extended path*. For more information, see *The Version-Extended Path* on page 58.

For example:

cleartool merge –graphical –to prog.c prog.c@@/main/LATEST

Using the **–graphical** option starts Diff Merge. For information about using Diff Merge, see Help in Diff Merge.

After merging, save the results and check in the version from the view (see *To Check In Files* on page 43).

Under the Hood: Checking In Files

The steps ClearCase follows when you issue the **checkin** command vary depending on the kind of view you use.

Checking In From a Snapshot View

When you issue a **checkin** command from a snapshot view, ClearCase handles the request as follows:

1. It copies your modifications to the VOB as a new version.

   The version you check in remains in the view, regardless of the config spec of the view.

2. It removes write permission for the file.

For any other instance of a hard-linked file loaded into a snapshot view, ClearCase copies the new version from the VOB into your view. (If your load rules specify a hard-linked element that appears in more than one VOB location, the element is copied into each of the appropriate locations in the directory tree of your view.)

Checking In From a Dynamic View

When you issue the **checkin** command from a dynamic view, ClearCase handles the request as follows:

1. It copies your modifications to the VOB as a new version.
2. It uses the version-selection rules in the config spec to select a version from the VOB. If the config spec selects a version other than the one you checked in, ClearCase displays a message. ClearCase may select another version if, for example, your view selects versions based on labels or time rules.
3. It removes the view-private file and provides transparent access to the version checked in to the VOB.

Checking In Elements in a VOB Enabled for ClearQuest

If you use the base ClearCase-ClearQuest integration (see *Checking Out Elements in a VOB Enabled for ClearQuest* on page 34), the version you are checking in must be associated with at least one change request; otherwise, the checkin cannot proceed. When you check in the version, the base ClearCase-ClearQuest integration displays

those change-request IDs whose associations you made during checkout. You can do the following:

- Keep the same change-request IDs.
- Delete some or all of the change-request IDs.
- Add new change-request IDs.

The base ClearCase-ClearQuest integration creates associations for new change-request IDs that you add, removes associations for change-request IDs that you delete, and updates information on existing ones.

Using the Association Dialog Window

If you use File Browser, the Association dialog window appears. You can do the following:

- Click **Browse** to navigate to and run another query.

  In the result set, select entries and click **Associate** to add the selections to **Associated Records**.

- In **Associated Records**, select an entry and click **Disassociate** to delete the current association.

In the Association dialog window, click **OK** to continue the checkin. For more information, click **Help**.

Using the Command Line Interface Options

If you use a **cleartool** command line interface (CLI), you can do the following with the options in Table 1 on page 36:

- Use the QUERY option to see a list of all open change-request IDs currently assigned to you.
- Use the REVIEW option to list and possibly delete current associations.
- Use the TYPEIN option to enter one or more change-request IDs with which to associate the version that you are checking in.

If the associations are correct, use the OK option to continue the checkin.

View the Versions for a Change Request from ClearQuest

To view the versions associated with a ClearQuest change request:

1. In ClearQuest, run a query to find the desired set of change requests.
2. In the query result set, select a change request to display its Record form.

3 On the Record form, click the **ClearCase** tab.

The **ClearCase** tab shows the last known names of the versions of ClearCase elements associated with the change request. Files without extended paths have not yet been checked in.

Updating a Snapshot View

4

The rules in the *config spec* of your view are usually designed to select a discrete set of versions from the VOB. For example, your view is usually intended to contain a set of versions that build successfully. However, when other developers check in new versions from their views, a snapshot view may become out of date or inconsistent with the versions in the VOB. To make sure that your view contains the set of versions the config spec selects, you must update it.

This chapter explains

- Starting an update operation
- What happens when you update a view
- Unloading elements

An update operation copies versions of elements from a VOB to your view. Only the checkin operation copies changes from your view back to a VOB.

Starting an Update Operation

You can start an update operation for

- The entire view
- At least one file or at least one directory tree

Updating the Entire View

Update the entire view periodically to make sure you have the correct version of all loaded files and directories.

To update the view, use **cleartool update** with any of the following options:

cleartool update [**–print**] [**cti·me** | **–pti·me**] *snapshot-view-path*

The *snapshot-view-path* argument is optional if you enter the **update** command from the root directory of the view.

Use these command options as follows. (For a description of all available command options, see the **update** reference page in the *Command Reference*.)

–print

Produces a preview of the update operation: instead of copying or removing files, **update** prints a report to standard output of the actions it would take for each specified element.

–cti·me

Sets the time stamp of a file element to the current time, that is, the time at which the version is copied into the view.The command option –**ctime** has no effect on directories (directories always use the current time). The initial default for the time stamp is set by the **mkview** command. Thereafter, the most recently used time scheme is retained as part of the state of the view and is used as the default behavior for the next update.

–pti·me

Sets the time stamp of a file element to the time at which the version was checked in to the VOB. The command option –**ptime** has no effect on directories. (Directories always use the current time.)

For example:

cleartool update ~/pat_1.4_cropcircle_sv

Note: You can use the Update Tool instead of the command line to update the view.

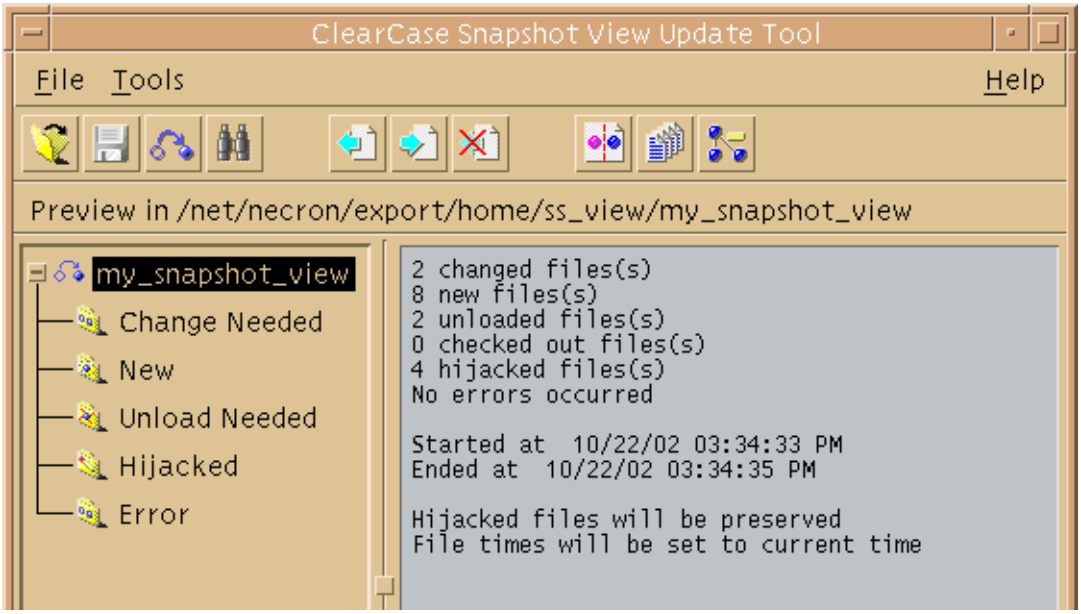
To Start the Update Tool

To start the Update Tool, enter the following command:

cleartool update –graphical *snapshot-view-path*

The *snapshot-view-path* argument is optional if you enter the **update** command from the root directory of the view. When the update is complete, the Update Tool window displays a categorized list of the actions taken to update the view (see Figure 7). For a description of this window, see the Help.

Figure 7 Update Tool Window



Updating Files and Directory Trees

To save time, you can update individual files or directories. (Rational ClearCase updates directories recursively.) Updating only specific parts of your view may eventually cause the view to contain an inconsistent set of versions.

Enter the following command:

cleartool update [**–print**] [**cti·me** | **–pti·me**] *paths-of-loaded-elements*

For information about these command options, see *Updating the Entire View* on page 49. For information about all available command options, see the **update** reference page in the *Command Reference*.

Note: You cannot update a checked-out file. To undo changes to a checked-out file and start over with the version in the VOB, cancel the checkout. See *Canceling Checkouts* on page 40.

Under the Hood: What Happens When You Update a View

When you start an update operation, ClearCase compares the version of the elements loaded in the view with the version the config spec selects in the VOB. If the config spec selects a version in the VOB that is different from the version loaded in your view, ClearCase copies the version from the VOB into your view (Figure 8). ClearCase does

not make this comparison or otherwise modify versions currently checked out to the view.

The update operation takes into account the fact that changes may be occurring in the VOB during the update. As ClearCase updates your view, other developers may check in new versions of elements that the load rules of your view select. To avoid loading an inconsistent set of versions, the update operation ignores versions in the VOB that meet both of the following conditions:

- The version was checked in after the moment the update began.
- The version is now selected by a config spec rule that involves the **LATEST** version label.

The update operation adjusts for the possibility that the system clocks on different hosts in a network may be out of sync (clock skew).

Unloading Elements

If the config spec of a view no longer selects an element, ClearCase removes, or unloads, it from the view. Unloading does not affect view-private files or view-private directories.

Updating can cause an element to be unloaded from a view in the following situations:

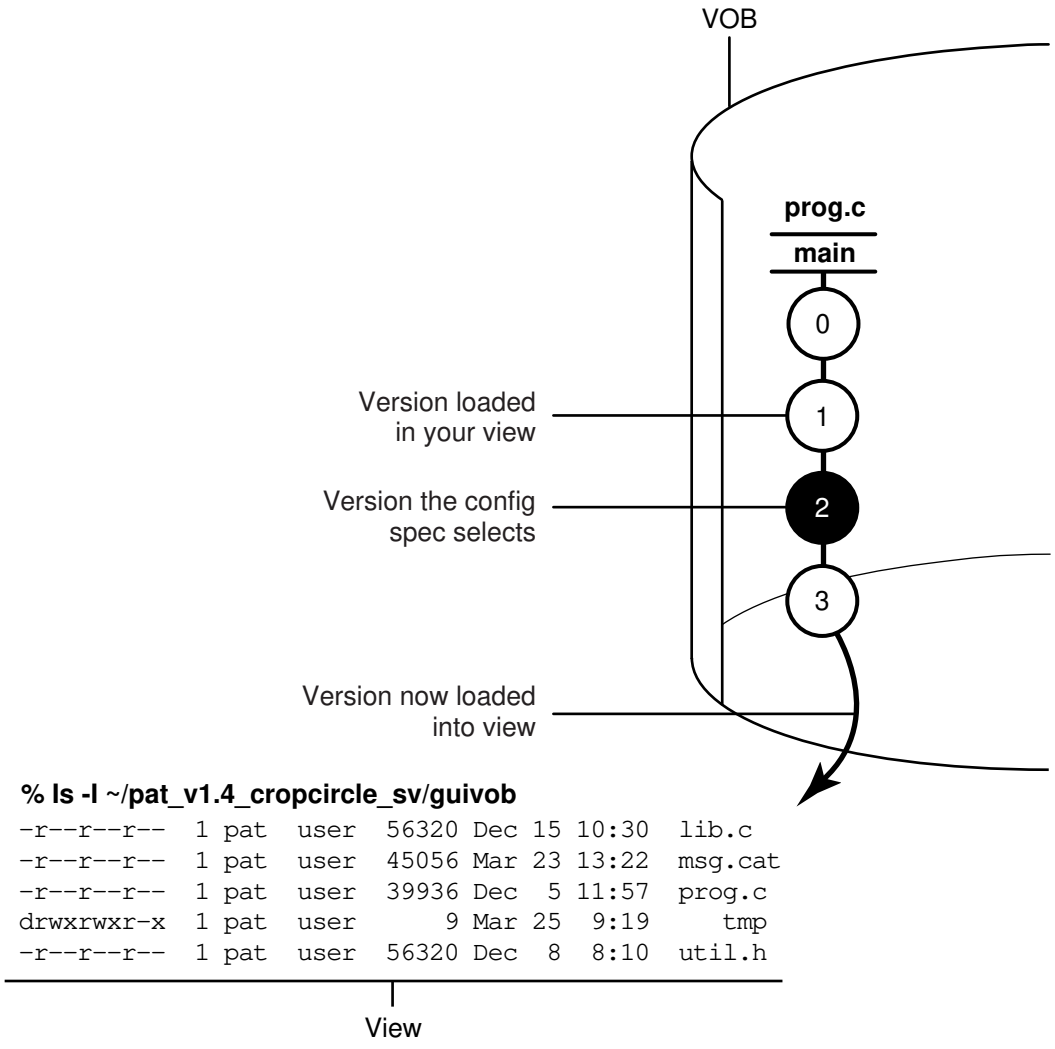
- You remove the load rule that specifies the element (or that specifies a directory element somewhere above it). For information about removing load rules, see *Adding or Modifying Load Rules* on page 20.
- The version-selection rules no longer select any version of the element. This can happen when your config spec selects a version of the parent directory that no longer contains a version of the file element.

Unloading Files

The action that ClearCase takes to unload a file depends on the current state of the file:

- For a file that is not checked out, ClearCase deletes the file from the view.
- For a *hijacked* file, ClearCase appends .unloaded to the file name, unless you use **update –overwrite** to delete hijacked files.
- For a checked-out file, ClearCase appends .unloaded to the file name. The version remains checked out to your view.

Figure 8 Update Operation



Unloading Directories

ClearCase unloads directories recursively. To unload a directory element, ClearCase unloads the files in the directory. If any view-private objects, hijacked files, or checked-out files are in the directory, or if the directory is currently in use (for example, if your current working directory is in or below the directory) ClearCase appends .unloaded to the name of the directory. For example, if the directory src contains view-private files, ClearCase renames the directory to src.unloaded.

Working On a Team

5

The development cycle presented so far is a fairly simple one in which everyone in an organization contributes to the same development project. But a software development cycle often involves several concurrent development projects. For example:

- You may want to experiment with some changes to the graphic user interface as a result of feedback from usability testing, but are not yet sure whether to include your changes in official builds.
- Another team may try to optimize the database schema without upsetting the current one.
- Another group may need to get a head start on a feature for the next release of the product.

This chapter describes the functions that Rational ClearCase provides to support parallel development. Parallel development is a style of working in which teams use the same set of source files for different, concurrent development projects:

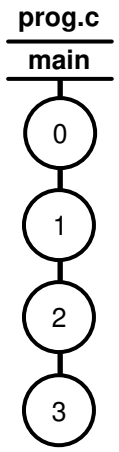
- Version trees
- Working on branches
- Merging
- Sharing control of a branch in an environment by using Rational ClearCase MultiSite

(You do not need to read the section about sharing control of a branch with developers at other sites unless your project manager or MultiSite administrator directs you.)

The Version Tree

Each time you revise and check in an element, ClearCase creates a new version of the element in the VOB. This linear progression is illustrated by Figure 9.

Figure 9 Linear Progression of Versions



ClearCase can organize the different versions of an element in a VOB into a version tree. Like any tree, a version tree has branches. Each branch represents an independent line of development. Changes on one branch do not affect other branches unless you merge. In Figure 10, **main**, **pat_usability**, and **db_optimize** are branches being used to develop different releases of the file element prog.c concurrently.

Under the Hood: The Initial Version on a Subbranch

When you create a subbranch for an element, which is any branch below the **main** branch, the initial version contains the same data as the version from which you start the branch (Figure 11). (The initial version on the **main** branch contains no data. For more information, see *Excluding Elements from Loading* on page 21.)

Figure 10 Version Tree of a File Element

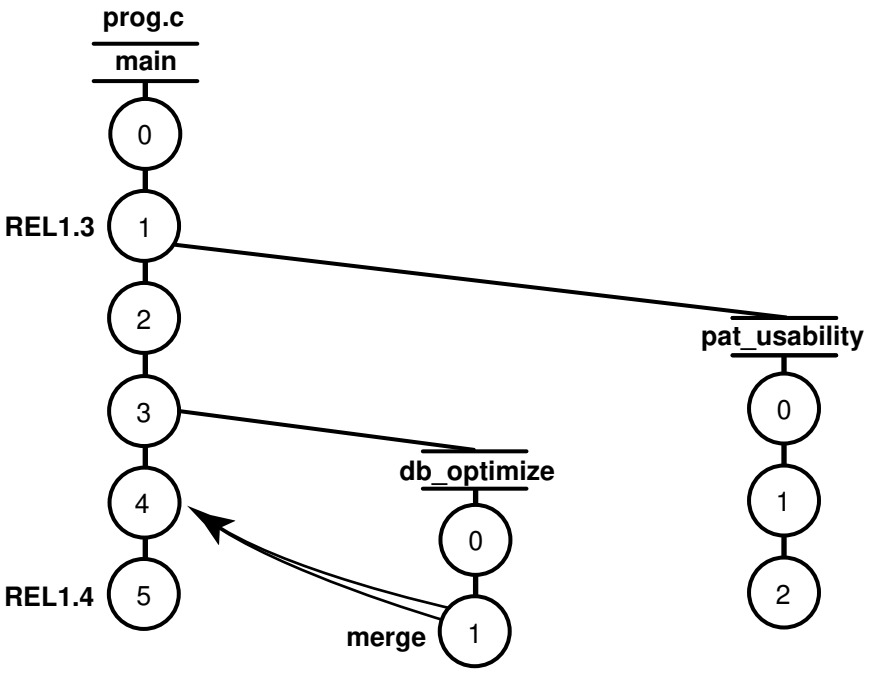
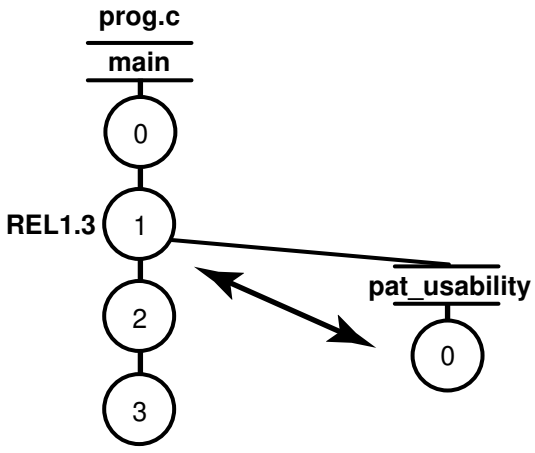


Figure 11 The Initial Version on a Subbranch



Working on Branches

The policies of your organization may dictate that each development project use its own branch to isolate its changes from other development projects. To adhere to this policy, each member of a project team uses a view whose config spec specifies the following information:

- The versions to select in the specific branch of the development project. As Figure 12 illustrates, some or all source files for the project have at least one version on the specified branch. If an element does not have a version on the specified branch, other rules in the config spec select a version of the element. In Figure 12, because lib.c does not have a version on the **pat_usability** branch, the view selects the version on the **main** branch.
- A special *make branch* rule. When this view checks out a version, the make-branch rule creates the branch of the development project (if it does not already exist).

For example, each member of the project team that is optimizing the database schema uses a view that selects versions on the **db_optimize** branch and creates new versions on that branch.

For more information about branches, see *Managing Software Projects* and the **mkbranch** reference page in the *Command Reference*.

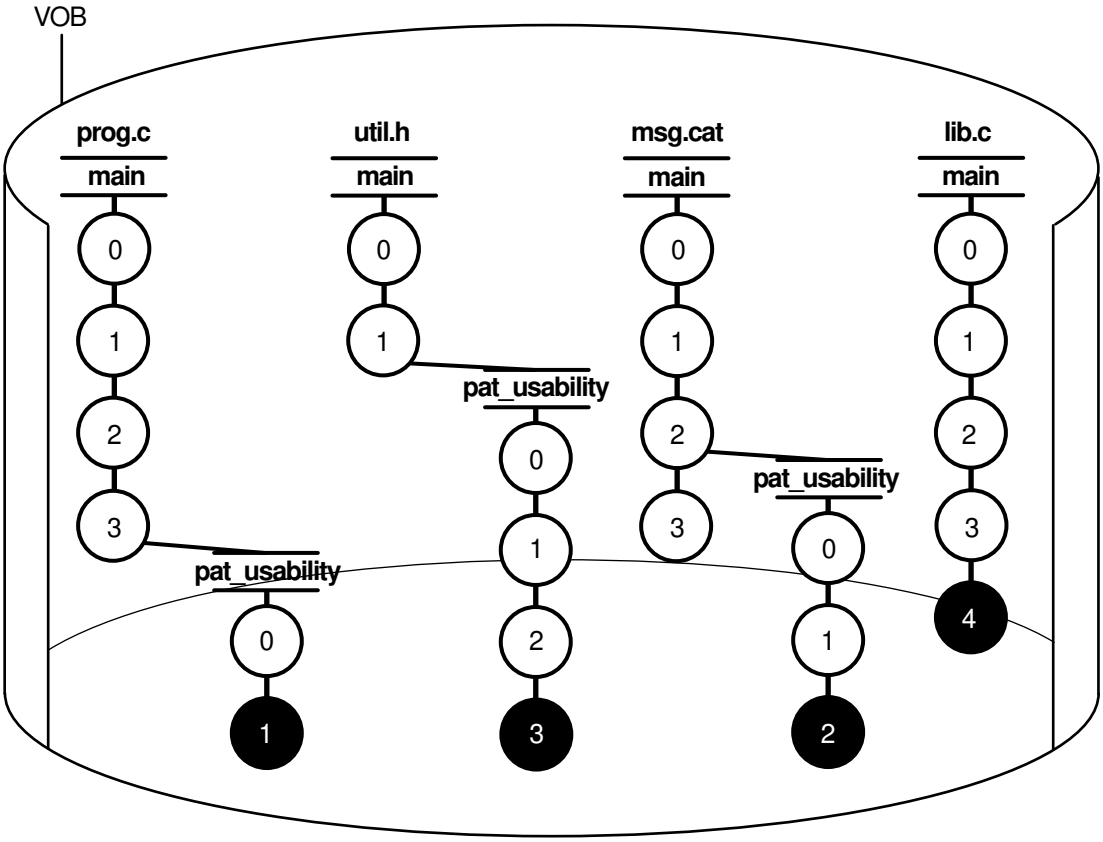
The Version-Extended Path

ClearCase commands and documentation use a notation to specify a version of an element on a branch. For example, util.h@@/main/2 specifies version 2 of util.h on the **main** branch; util.h@@/main/r1_bugs/bug404/1 specifies version 1 of util.h on the **bug404** subbranch below the **r1_bugs** subbranch, which is below the **main** branch (Figure 13).

From a command-line interface, you can use version-extended paths to access versions other than the ones currently selected by your view. To view the contents of a version that is not currently in a snapshot view, you must use the **cleartool get** command in addition to version-extended paths.

For information about the syntax for version-extended paths, see the **pathnames_ccase** reference page in the *Command Reference*.

Figure 12 Elements Have Common Branches



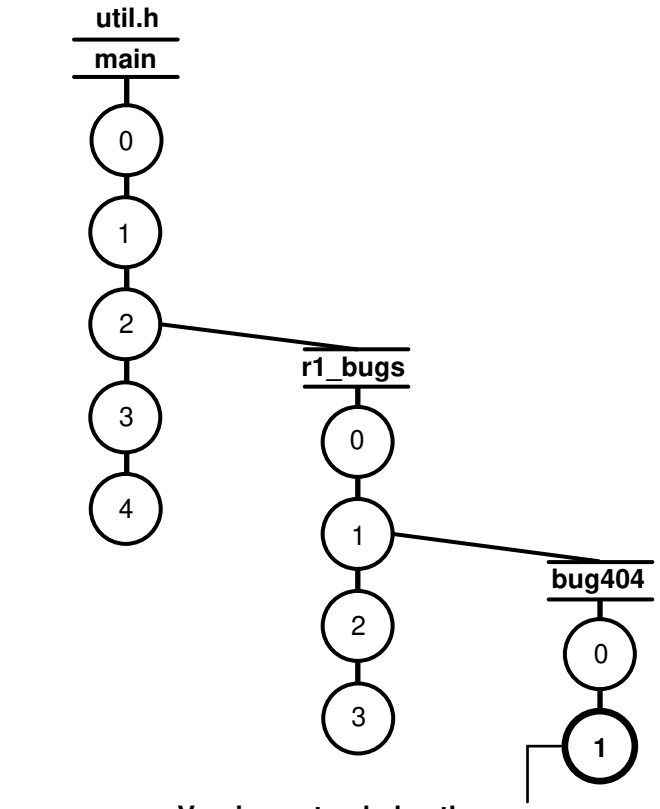
Load Rules

```
load  /guivob/prog.c
load  /guivob/util.h
load  /guivob/msg.cat
load  /guivob/lib.c
```

Version-selection Rules

```
element * /main/pat_usability/LATEST
element * /main/LATEST -mkbranch pat_usability
```

Figure 13 Version-Extended Paths



Merging

In a parallel development environment, the opposite of branching is merging. In the simplest scenario, merging incorporates changes on a subbranch into the **main** branch. However, you can merge work from any branch to any other branch. ClearCase includes automated merge facilities for handling almost any scenario.

One of the most powerful of ClearCase features is source control of directories. Each version of a directory element catalogs a set of file elements, directory elements, and VOB symbolic links. In a parallel development environment, directory-level changes may occur as frequently as file-level changes. All the merge scenarios discussed in this chapter apply to both directory and file elements.

This section describes the following merge scenarios:

- Merging all changes made on a single subbranch (see *Scenario: Merging All Changes Made on a Subbranch* on page 67)
- Merging selectively from a single subbranch (see *Scenario: Selective Merge from a Subbranch* on page 68)
- Removing the contributions of some versions on a single subbranch (see *Scenario: Removing the Contributions of Some Versions* on page 71)
- Recording merges that occur outside ClearCase (see *Recording Merges That Occur Outside ClearCase* on page 72)

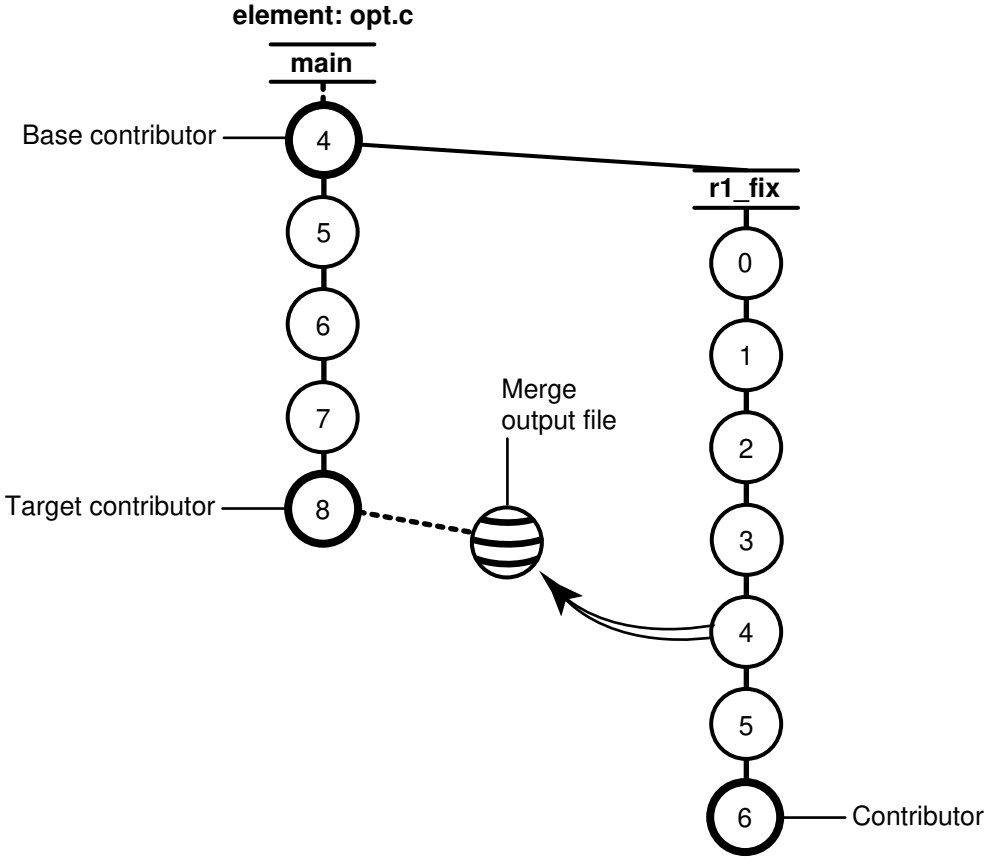
ClearCase also supports merging work from many branches to a single branch; this is typically a project manager's or integrator's task (see *Managing Software Projects*).

Under the Hood: How ClearCase Merges Files and Directories

A merge combines the contents of two or more files or directories into a new file or directory. The merge algorithm uses the following files during a merge (Figure 14):

- Contributors, which are typically one version from each branch you are merging. (You can merge up to 15 contributors.) You specify which versions are contributors.
- The base contributor, which is typically the closest common ancestor of the contributors. (For selective merges, subtractive merges, and merges in an environment with complex branch structures, the base contributor may not be the closest common ancestor.) If all the contributors are versions of the same element, ClearCase determines which contributor is the base contributor (but you can specify a different one). For more information about determining a base contributor, see *Determination of the Base Contributor* on page 64.
- The target contributor, which is typically the latest version on the branch that will contain the results of the merge. You determine which contributor is the target contributor.

Figure 14 Versions Involved in a Typical Merge



- The merge output file, which contains the results of the merge and is usually checked in as a successor to the target contributor. By default, the merge output file is the checked-out version of the target contributor, but you can choose a different file to contain the merge output.

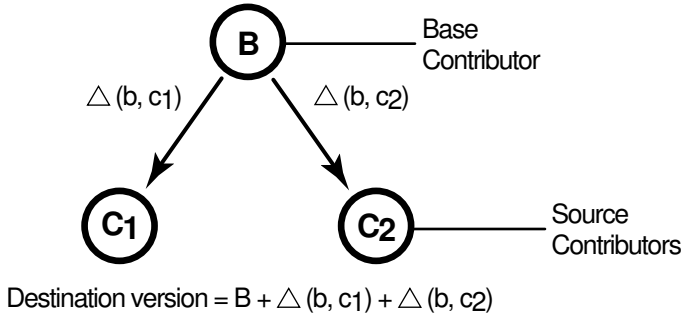
To merge files and directories, ClearCase takes the following steps:

1. It identifies the base contributor.
2. It compares each contributor against the base contributor (Figure 15).
3. It copies any line that is unchanged between the base contributor and any other contributor to the merge output file.
4. For any line that has changed between the base contributor and one other contributor, ClearCase accepts the change in the contributor; depending on how you started the merge operation, ClearCase may copy the change to the merge output file. However, you can disable the automated merge capability for any

given merge operation. If you disable this capability, you must approve each change to the merge output file.

5 For any line that has changed between the base contributor and more than one other contributor, ClearCase requires that you resolve the conflicting difference.

Figure 15 ClearCase Merge Algorithm



File Merge Algorithm

A merge is a straightforward extension of a file comparison. Instead of displaying the differences, Diff Merge analyzes them (sometimes with your help) and copies sections of text to the output file:

- Sections in which there are no differences among the contributors are copied to the output file.
- When *one* contributor differs from the base contributor, Diff Merge accepts the change and copies the modified section of the contributor to the output file:

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

(You can turn off automatic acceptance of this kind of change.)

- When two or more contributors differ from the base contributor, Diff Merge detects the conflict, and prompts you to resolve it. It displays all contributor differences and allows you to accept or reject each one.

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

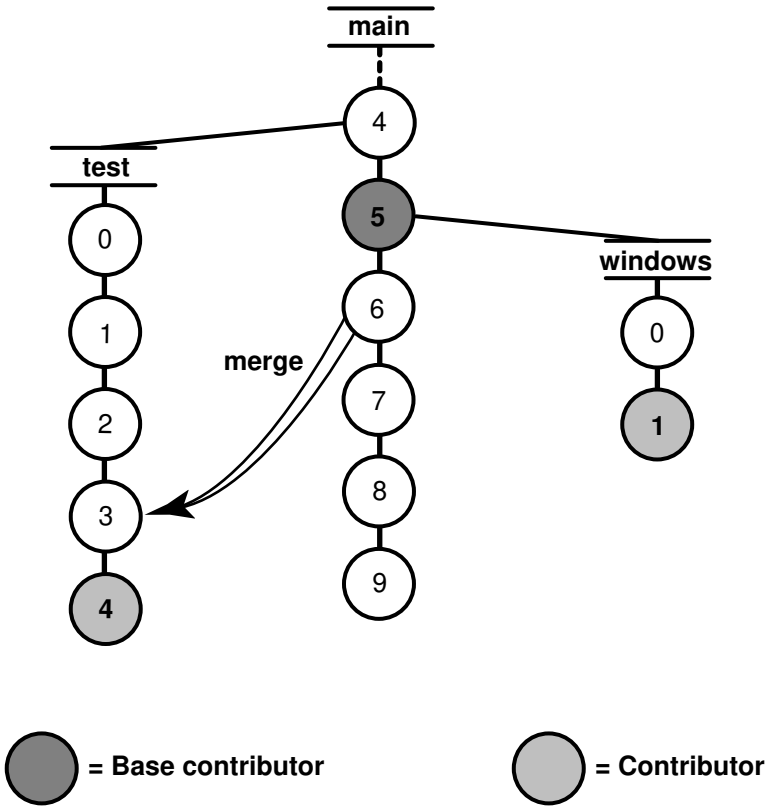
Be sure to verify that the changes you accept produce consistent merged output. For example, after performing a merge involving file util.h, you can compare files util.h.contrib (which contains its previous contents) and the new util.h (which contains the merge output).

Determination of the Base Contributor

If all the contributors are versions of the same element, Diff Merge determines the base contributor automatically. It examines the version tree of the element, which includes all the merge arrows created by previous merge operations. This examination reveals the relationships among versions from the standpoint of their contents (which versions contributed their data to this version?), rather than their creation order (which versions were created before this version?). Diff Merge selects as the base contributor the closest common ancestor in this enhanced version tree.

Figure 16 illustrates a common case of merging. If no merges have been performed in the element, the actual common ancestor (**5**) of the contributors (**test/4** and **windows/1**) in the version tree is selected to be the base contributor.

Figure 16 Determination of the Base Contributor for a Merge



If the contributors are not all versions of the same element, there is no common ancestor (or other base contributor). In this case, ClearCase turns off automated merging, and you must resolve all discrepancies among the contributors.

Recording of Merge Arrows

Under the following conditions, the merge is recorded by creating one or more merge arrows (hyperlinks of type **Merge**):

- All contributor files must be versions of the same file element.
- One of the contributors must be a checked-out version, and you must specify this version as the target to be overwritten with the merge output (the **–to** option in the **merge** command). (Alternatively, you can optionally create merge arrows without performing a merge; in this case, you do not need to check out any of the contributors.)

- You must not perform the merge but suppress creation of merge arrows.
- You must not use any of these options: selective merge, subtractive merge, or base contributor specification (the **–insert**, **–delete**, and **–base** options in the **merge** command).

Diff Merge draws an arrow from each contributor version (except the base contributor) to the target version. You can see merge arrows with the Version Tree Browser.

Locating Versions with Merge Hyperlinks

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

cleartool describe util.h@@/main/3

```
version "util.h@@/main/3"
.
.
.
 Hyperlinks:
 Merge@278@/vob_3 /vob_3/src/util.h@@/main/rel2_bugfix/1
 -> /vob_3/src/util.h@@/main/3
```

Directory Merge Algorithm

Each version of a ClearCase directory element contains the names of certain file elements, directory elements, and VOB symbolic links. Diff Merge can process two or more versions of the same directory element, producing a directory version that reflects the contents of all contributors. The algorithm is similar to that for a file merge. Diff Merge prompts for user interaction only when two or more of the contributors are in conflict.

One of the directory versions—the merge target—must be checked out. (Typically, it is the version in your view.) Diff Merge updates the checked-out directory by adding, removing, and changing names of elements and links.

> **Note**: A directory merge does not leave behind a .contrib file, with the pre-merge contents of the target version.

Merging Directories

We recommend that you use this procedure when merging directories:

1. Ensure that all contributor versions of the directory are checked in.
2. Check out the target version of the directory.

3 Perform the directory merge immediately, without making any other changes to the checked-out version.

If you follow this procedure, it easier to determine exactly what the merge accomplished. Enter a **diff –predecessor** command on the checked-out version, which has just been updated by **merge**.

Using ln and rmname in Diff Merge

ClearCase uses VOB hard links to implement directory merges. You can use the **ln** and **rmname** commands to perform full or partial merges manually. See the **ln** and **rmname** reference pages in the *Command Reference*.

Scenario: Merging All Changes Made on a Subbranch

Merging all changes made on a subbranch is the simplest and most common scenario (Figure 17).

Bug fixes for an element named opt.c are being made on branch **r1_fix**, which was created at the baseline version **RLS1.0** (**/main/4**). Now, all the changes made on the subbranch are to be incorporated into **main**, where a few new versions have been created in the meantime.

Task Overview

Merging the changes from the **r1_fix** branch involves the following tasks:

1 Access your dynamic view (see *In a Dynamic View* on page 29) or snapshot view (see *In a Snapshot View* on page 29).

The view must select the target version, which, in Figure 17, is opt.c@@/main/8.

2 If the target version is checked out to your view for other revisions, create a pre-merge checkpoint by checking it in. To make it easier to find this checkpoint, consider labeling the version.

3 Use the Merge Manager or **cleartool findmerge –merge** –**gmerge** to find elements with versions on a specific subbranch and automatically merge any nonconflicting differences. For example, in Figure 17, you find elements with versions on the **r1_fix** subbranch. To start the Merge Manager, enter the following command:

clearmrgman

In your project, several elements might have versions on the **r1_fix** branch. With the Merge Manager, you can choose for which elements you merge changes from one branch to another.

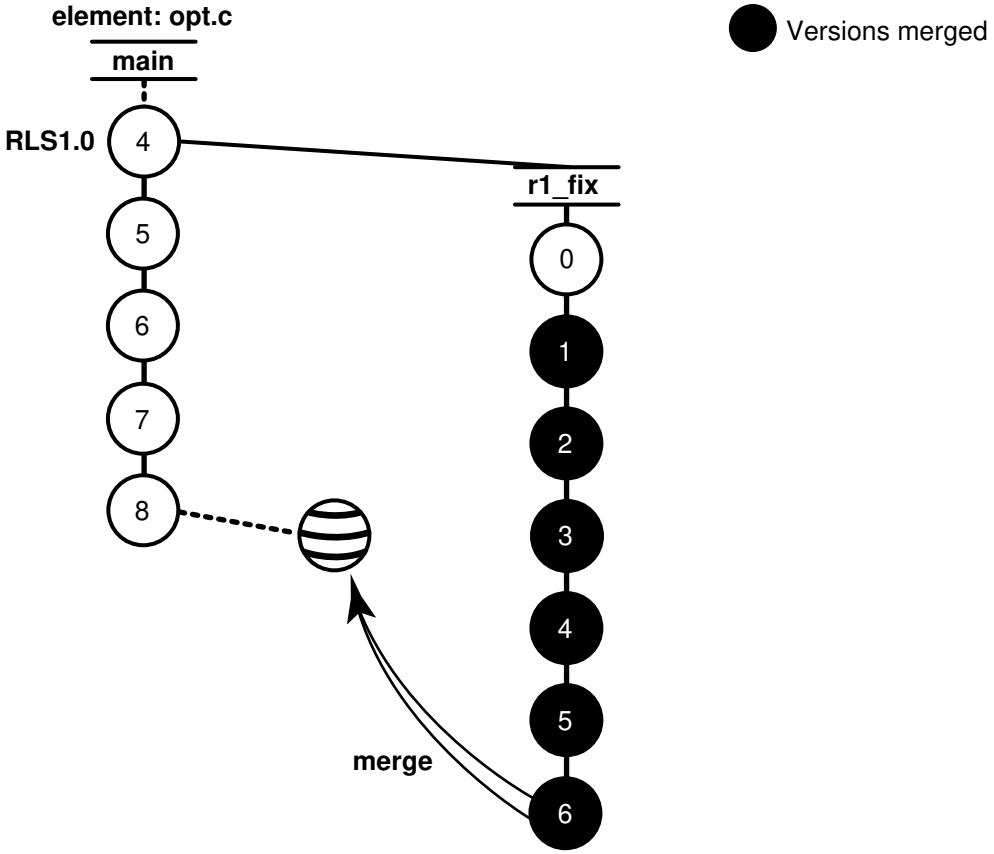
4 Use Diff Merge to resolve any conflicting differences between merge contributors.

5 Test the merge results in the view you set in Step 1. Then check in the target version (which contains the results of the merge).

Getting More Information

For detailed information about completing this task, see the **findmerge** reference page in the *Command Reference* or ClearCase Help. For information about starting Help, see *To Access Online Information* on page 1.

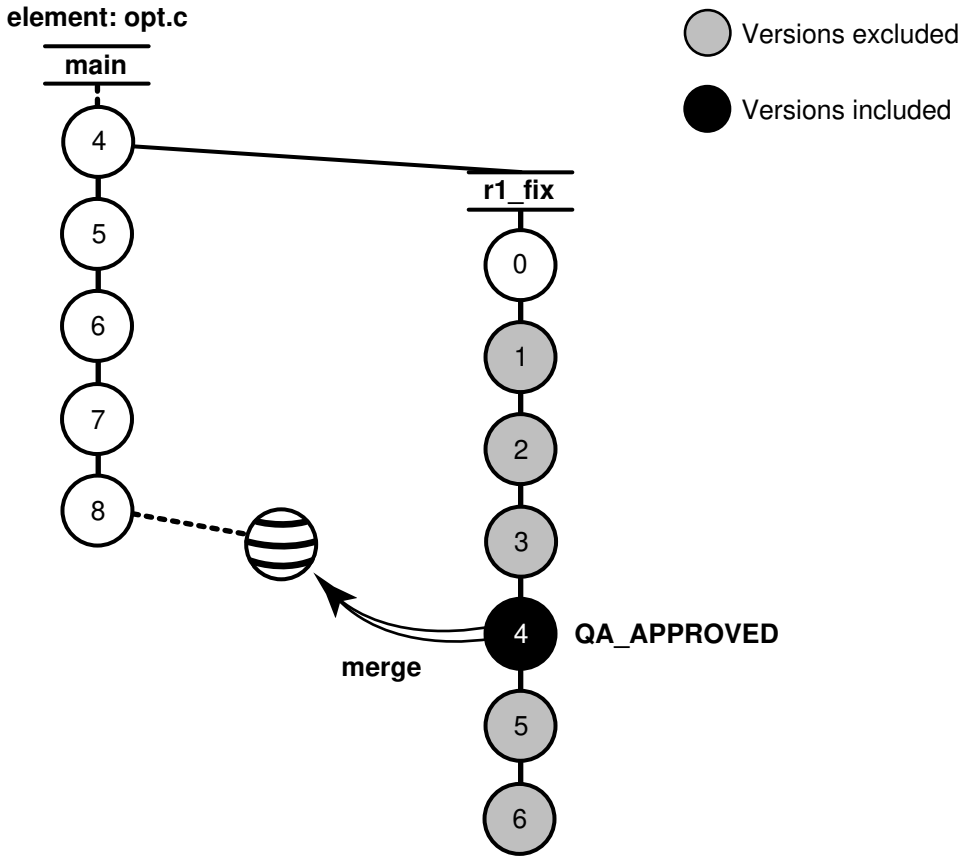
Figure 17 Merging All Changes from a Subbranch



Scenario: Selective Merge from a Subbranch

In the selective merge scenario, the project manager wants to incorporate into new development several lines of code that were added in version **/main/r1_fix/4** (Figure 18).

Figure 18 Selective Merge from a Subbranch



It is critical that you merge only the lines of code as written in this version: it was used and verified to fix a specific bug that prevents further development on the new project.

Selective merges can be tricky: versions you exclude as contributors to the merge may contain needed revisions. For example, if the function you added in **/main/r1_fix/4** relies on a variable definition that was added in **/main/r1_fix/2**, you must include version 2 in the merge.

Merging a Range of Versions

You can also specify a single range of consecutive versions to contribute to the merge. For example, if you need the variable definitions added in **/main/r1_fix/2** as well as the code added in **/main/r1_fix/4**, you can include versions 2 through 4 in the merge.

Task Overview

Merging selective versions from the **r1_fix** branch involves the following tasks:

1. Access your dynamic view (see *In a Dynamic View* on page 29) or snapshot view (see *In a Snapshot View* on page 29).

   The view must select the target version, which in Figure 18 is opt.c@@/main/8.

2. If the target version is checked out to your view for other revisions, create a pre-merge checkpoint by checking it in.

3. To determine which versions contain changes that you want to merge to the target version, use the Version Tree Browser and the History Browser. In a snapshot view, use the **cleartool get** command to see the contents of versions not loaded into your view. (For information about opening a version not currently in your view, see *To Compare with a Version in a Dynamic View* on page 39 or *Accessing Elements Not Loaded into a Snapshot View* on page 86.)

4. To start the merge, check out the target version, and then issue the **cleartool merge** command with the **–insert –graphical** arguments. (You cannot start a selective merge from Diff Merge.)

   For example, the following commands merge only the changes in version 4 on the **r1_fix** branch:

   cleartool checkout opt.c
   cleartool merge –graphical –to opt.c –insert –version /main/r1_fix/4

   These commands merge only the changes in versions 2 through 4 on the **r1_fix** branch:

   cleartool checkout opt.c
   cleartool merge –graphical –to opt.c –insert –version
   /main/r1_fix/2 /main/r1_fix/4

5. In Diff Merge, complete the merge. Then save the results and exit. For information about using Diff Merge, see the Help.

6. Test the merge results in the view you set in Step 1. Then check in the target version.

   Note: In a selective merge, ClearCase does not create a merge arrow. A merge arrow indicates that all the data of a version has been merged, not parts of it.

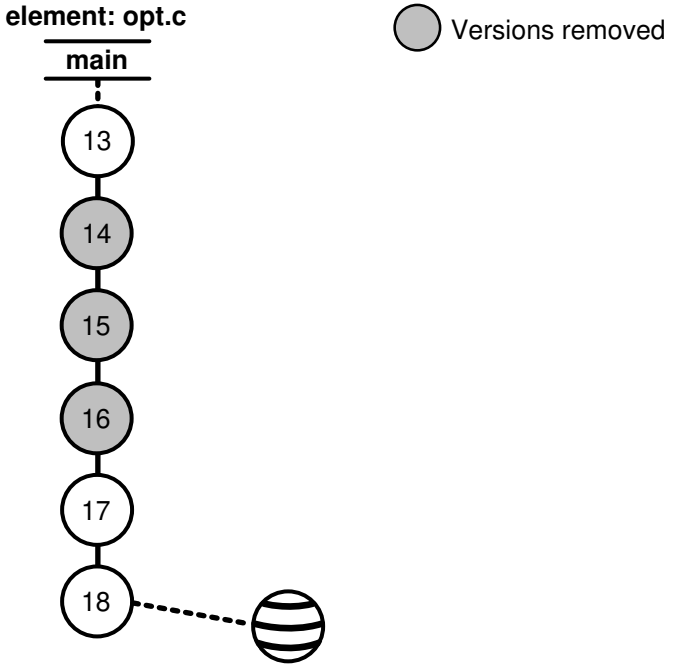
Getting More Information

For detailed information about completing this task, see the **merge** and **version_selector** reference pages in the *Command Reference* or see Help. For information about starting Help, see *To Access Online Information* on page 1.

Scenario: Removing the Contributions of Some Versions

The project manager has decided that a new feature, implemented in versions 14 through 16 on the **main** branch, will not be included in the product. You must perform a subtractive merge to remove the changes made in those versions (Figure 19).

Figure 19 Removing the Contributions of Some Versions



Task Overview

A subtractive merge is the opposite of a selective merge: it removes from the checked-out version the changes made in one or more of its predecessors. Performing a subtractive merge involves the following tasks:

1 Access your dynamic view (see *In a Dynamic View* on page 29) or snapshot view (see *In a Snapshot View* on page 29).

The view must select the branch from which you want to remove revisions.

2 If the target version is checked out to your view for other revisions, create a pre-merge checkpoint by checking it in. In Figure 19, the target version is opt.c@@/main/18.

3 To determine which versions contain changes you want to remove, use the Version Tree Browser and the History Browser. From a snapshot view, use the **cleartool get** command to see the contents of versions not loaded into your view. (For information about opening a version not currently in your view, see *To Compare with a Version in a Dynamic View* on page 39 or *Accessing Elements Not Loaded into a Snapshot View* on page 86.)

4 To perform the merge, check out the target version, and then use the **cleartool merge** command with the **–delete –graphical** arguments. (You cannot start a subtractive merge from Diff Merge.)

For example, the following commands remove revisions to versions 14 through 16 on the **main** branch:

```
cleartool checkout opt.c
cleartool merge –graphical –to opt.c –delete –version  /main/14 /main/16
```

5 In Diff Merge, complete the merge. Then save the results and exit. For information about using Diff Merge, see the Help.

6 Test the merge results in your view. Then check in the target version (which contains the results of the merge).

Note: In a subtractive merge, ClearCase does not create a merge arrow. A merge arrow indicates that data has been merged, not removed.

Getting More Information

For detailed information about completing this task, see the **merge** and **version_selector** reference pages in the *Command Reference* or see Help. For information about starting Help, see *To Access Online Information* on page 1.

Recording Merges That Occur Outside ClearCase

You can merge versions of an element manually or with any available analysis and editing tools. To update the version tree of an element with a merge that occurs outside ClearCase, do the following:

1 Check out the target version.

2 Perform the merge with your own tools.

3 Check in the target version.

4 Then record the merge by drawing a merge arrow from the contributors to the new version that contains the result of the merge.

After you draw the merge arrow, your merge is indistinguishable from one performed with ClearCase tools.

For example, use the following commands to merge a version of nextwhat.c on the **enhance** branch to the branch currently selected by your view:

cleartool checkout nextwhat.c
```
Checkout comments for "nextwhat.c":
```
merge enhance branch
.
```
Checked out "nextwhat.c" from version "/main/1".
```

<*use your own tools to merge data into checked-out version***>**

cleartool merge –to nextwhat.c –ndata –version .../enhance/LATEST
```
Recorded merge of "nextwhat.c".
```

The **–ndata** option suppresses the merge but creates merge arrows as if ClearCase had merged the versions.

Getting More Information

For detailed information about completing this task, see the **merge** and **version_selector** reference pages in the *Command Reference* or see Help. For information about starting Help, see *To Access Online Information* on page 1.

Sharing Control of a Branch with Developers at Other Sites

> **Note**: This section describes how to request control of a branch from another development site. Do not read this section unless your project manager or MultiSite administrator directs you to.

If your company uses MultiSite to distribute development among multiple geographical sites, you may share source files with developers at other sites. Each site has its own replica of the VOB, and developers work in their site-specific replica (known as the *current replica*). Each replica controls (masters) a particular branch of an element, and only developers at the site of that replica can work on that branch. In this scenario, MultiSite branch mastership does not affect you, and you can do your work as usual.

However, sometimes elements cannot have multiple branches. For example, some file types cannot be merged, so development must occur on a single branch. In this scenario, all developers must work on a single branch (usually, the **main** branch). MultiSite allows only one replica to master a branch at any given time. Therefore, if a developer at another site needs to work on the element, she must request mastership of the branch.

> **Note**: The developer can also request mastership of branch types. For more information, see the *Administrator's Guide* for Rational ClearCase MultiSite.

For example, the file doc_info.doc cannot be merged because it is a file type for which you do not have a *type manager*, but developers at different sites need to make changes to it. If the branch is mastered by your current replica, you can check out the file. If the branch is mastered by another replica, you cannot check out the file. If you try to check out the file, ClearCase presents an error message:

cleartool checkout –c "command changes" doc_info.doc

```
cleartool: Error: Cannot checkout branch "/main".
The branch is mastered by replica "sanfran_hub".
Current replica is "boston_hub".
cleartool: Error: Unable to check out "doc_info.doc".
```

For you to check out the file reserved or to check in the file after a nonmastered checkout, your current replica must master the branch. You can request mastership with a **cleartool** command.

If you have permission to request mastership from the master replica of the branch, if mastership requests are enabled, and if there are no blocking conditions, then the mastership change is made at the master replica, and a MultiSite update packet that contains the change is sent to your current replica. When your current replica imports the packet, it receives mastership of the branch and you can check out the file.

> **Note**: Authorizing developers to request mastership and enabling mastership requests at a replica are tasks performed by the MultiSite administrator. For more information, see the *Administrator's Guide* for Rational ClearCase MultiSite.

When you use mastership requests to transfer control of a branch, you can use either of two methods to do your work:

- Request mastership of the branch and wait for mastership to be transferred to your current replica; then perform a reserved checkout. You must use this method if you cannot or do not want to merge versions of the element.
- Request mastership of the branch and use a nonmastered checkout to check out the branch immediately. You may have to perform a merge before you can check in your work.

The following sections describe both methods.

To Request Mastership of a Branch and Wait for the Transfer

1 At a command prompt, enter a **cleartool reqmaster** command for the branch you need to check out.

cleartool reqmaster –c "add info re new operating systems" read_me_first.doc@@/main

```
read_me_first.doc@@/main: Change of mastership at sibling replica
"sanfran_hub" was successful.
Mastership is in transit to the new master replica.
```

2 Wait for mastership to be transferred to your current replica. To list the master replica of a branch, use **describe**:

cleartool describe read_me_first.doc@@/main

```
branch "read_me_first.doc@@/main"
  created 15-May-99.13:32:05 by sg.user
  branch type: main
  master replica: boston_hub@/doc
...
```

In this example, your current replica is **boston_hub** in the VOB family **/doc**. The output of the **describe** command shows that **boston_hub** is the master replica of the branch, which means that you can check out the branch as reserved.

3 Perform a reserved checkout, edit the file, and check in your work.

To Check Out the Branch Before Mastership Is Transferred

If you can merge versions of the element you need to check out, you can work on the file while you wait for mastership to be transferred to your replica.

To use this method from the command line:

1 Enter a **reqmaster** command for the branch you need to check out.

cleartool reqmaster –c "fix bug #28386" prog.c@@/main/integ

```
prog.c@@/main/integ: Change of mastership at sibling replica
"lib_lex" was successful.
Mastership is in transit to the new master replica.
```

2 Use **cleartool checkout –unreserved –nmaster** to perform a nonmastered checkout.

cleartool checkout –c "fix bug #28386" –unreserved –nmaster prog.c@@/main/integ

3 Make changes to the element.

4 Wait for mastership to be transferred to your current replica. To list the master replica of a branch, use **describe**:

cleartool describe /vobs/lib/prog.c@@/main

```
branch "/vobs/lib/prog.c@@/main"
  created 15-May-99.13:32:05 by nlg.user
  branch type: main
  master replica: lib_london@/vobs/lib
...
```

5 Check in the element. If the checkin succeeds, you are finished.

cleartool checkin –nc prog.c

```
Checked in "prog.c" version "/main/65".
```

If the checkin fails because you have to perform a merge, proceed to Step 6:

cleartool checkin –nc prog.c

```
cleartool: Error: The most recent version on branch "/main" is not
the predecessor of this version.
cleartool: Error: Unable to check in "prog.c".
```

6 Merge from the latest version on the branch to your checked-out version.

cleartool merge –to prog.c –version /main/LATEST

(if necessary, you are prompted to resolve conflicts)

```
Moved contributor "prog.c" to "prog.c.contrib".
Output of merge is in "prog.c".
Recorded merge of "prog.c".
```

7 Check in the element.

Troubleshooting

If the request for mastership fails because there are checkouts on the branch at the master replica, try your request again later or ask the other developer to check in the file or directory and then try again. If you receive other errors, contact your project manager or MultiSite administrator.

Other Tasks

6

Chapter 3, *Working in a View*, describes tasks you perform daily or weekly. You may need to perform some of these tasks less often:

- Adding files and directories to source control
- Moving, removing, and renaming elements
- Accessing elements not loaded into a snapshot view
- Registering a snapshot view
- Regenerating .ccase_svreg
- Moving views
- Regenerating the .view.dat file for a snapshot view
- Accessing views and VOBs across platform types

Adding Files and Directories to Source Control

You can add files or directories to source control at any time.

To Add Elements to Source Control

To add view-private files and directories to source control, or to make placeholders for nonexistent files and directories:

1. Go to the view used for your development task.

   The *version-selection rules* of your view determine the branch on which the first version of the element is created. Make sure your view creates versions on an appropriate *branch*.

2. Change to the parent directory under which you want to add files and directories to source control.

   For snapshot views, the path from which you add to source control does not need to be loaded. However, it must match the VOB namespace.

3. Check out the parent directory element by entering **cleartool checkout –nc** *directory-name*. We suggest using the **–nc** option because ClearCase appends appropriate comments when you modify directory elements.

4 Do **one** of the following:

- To add a directory to source control, enter this command:

  cleartool mkdir *directory-name*

- To add a file to source control, enter this command:

  cleartool mkelem *file-name*

- To make placeholders for nonexistent objects, enter one of these commands:

  cleartool mkdir *directory-element-path*
  cleartool mkelem *file-element-path*

By default, when you add an element, it remains checked out. When you finish modifying the new elements, check them in. Elements you add to a directory element are visible only in your view until you check in the directory.

For more information about the **mkelem** command, see *Under the Hood: What Happens When You Add to Source Control* and the **mkelem** reference page in the *Command Reference*.

Under the Hood: What Happens When You Add to Source Control

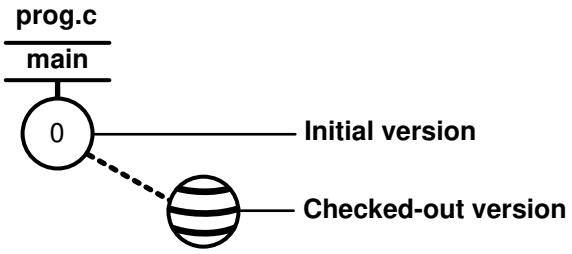
The **mkelem** command always creates an element and initializes its version tree by creating a single branch (named **main**) and a single, empty version (version 0) on that branch. The following arguments for the **mkelem** command determine optional ClearCase behavior:

- Using **mkelem** with no arguments checks out the element. Any view-private data that corresponds to the element path remains in your view only and is added to version 1 in the VOB when you check in (Figure 20).
- Using **mkelem –ci** checks in the element, using any existing view-private data that corresponds to the element path as the content for version 1. The config spec of your view determines the branch on which ClearCase creates version 1.
- Using **mkelem –nco** suppresses automatic checkout; **mkelem** creates the new element, along with the **main** branch and version**/main/0**, but does not check it out. If *element-path* exists, it is moved aside to a .keep file.
- (Replicated VOBs only) Using **mkelem –master** or **mkdir –master** assigns to your *current replica* mastership of all branches created during element creation. You will be able to create new versions on the branches.

  Using **mkelem** or **mkdir** without the **–master** option assigns mastership of a new branch to the VOB replica that masters the associated branch type. If this replica is not your current replica, you cannot create new versions on the branch.

Other views do not see the element until the parent directories of the element are checked in (you must do this yourself) and you check in the file or directory.

Figure 20 Creating an Element



File Types and Element Types

Each element is an instance of an element type. You can specify an element type with the **–eltype** option. (The **lstype –kind eltype** command lists the element types in a VOB.) The element type must already exist in the VOB in which you are creating the new element, or must exist in the administrative VOB hierarchy associated with the VOB in which you are creating the new element. A **mkelem –eltype directory** command is equivalent to a **mkdir** command.

If you do not specify an element type on the command line, **mkelem** uses the magic files to determine an element type with which to perform file-typing. This involves matching patterns in the name of the new element (and examining the existing view-private file with that name, if one exists; see *Conversion of View-Private Files to Elements* on page 80). If file-typing fails, an error occurs and no element is created:

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

For more information about file-typing, see **cc.magic** in the *Command Reference*.

Access Mode

Elements are controlled by ClearCase permissions, as described in the **permissions** reference page in the *Command Reference*, not by the access modes for files and directories that are set by the operating system. When your view selects a checked-in version of an element, all of its write permissions are turned off. When you check out an element, write permissions are added to the view-private copy. (For more information, see *Under the Hood: What Happens When You Check Out a File or Directory* on page 33.)

Because file elements are read-only, the mode to which your umask is applied is `444` (not `666`) for a file element. When you convert a view-private file to an element (see *Conversion of View-Private Files to Elements* on page 80), its read and execute rights become those of the new element.

Conversion of View-Private Files to Elements

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

- By default, **mkelem** creates an empty version 0 of the new element, then checks out the new element to your view. The view-private file becomes the checked-out version of the new element.
- If you use the **–ci** option (check in), **mkelem** does all of the above, then proceeds to check in version 1 of the new element. Thus, version 1 has the contents of the view-private file. With **–ptime**, **mkelem** preserves the modification time of the file being checked in.
- If you use the **–nco** option (no check out), **mkelem** creates an empty version 0 of the new element.

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

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

  Note: If **mkelem** does not complete correctly, your view-private file may be left under the .mkelem file name.

Conversion of Nonshareable Derived Objects to Elements

mkelem does not perform any special processing for a nonshareable derived object. The process is the same as for a shareable derived object, as described in *Conversion of View-Private Files to Elements* on page 80. However, when you check in version 1 of the

new element, the checkin converts the nonshareable derived object to a shareable derived object, then checks it in.

> **Note**: When a nonshareable derived object is converted to a shareable derived object, its derived object ID changes. For more information, see *Derived Objects and Configuration Records* in *Building Software*.

Creation of Directory Elements

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

Auto-Make-Branch During Element Creation

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

```
element * CHECKEDOUT
element * .../gopher_port/LATEST
element * V1.0.1 -mkbranch gopher_port
element * /main/0 -mkbranch gopher_port
```

The /main/0 rule in the last line allows you to create elements. In this case, a **gopher_port** branch is created for the new element, and this branch is checked out instead of **main**:

cleartool mkelem –c "new element for Gopher porting work" base.h

```
Created element "base.h" (type "text_file").
Created branch "gopher_port" from "base.h" version "\main\0".
Checked out "base.h" from version "\main\gopher_port\0".
```

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

Creation of Elements in Replicated VOBs

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

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

```
element * CHECKEDOUT
element * .../gms_dev/LATEST
element * /main/0 –mkbranch gms_dev
```

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

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

Element Object and Version References

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

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

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

cleartool mkattr BugNum 403 msg.c *(attaches attribute to version)*
cleartool mkattr BugNum 403 msg.c@@ *(attaches attribute to element)*

The first command operates on the version of the element selected in your view, but the second command operates on the element itself.

> **Caution**: Do not create elements whose names end with the extended-naming symbol. ClearCase software cannot handle such elements.

Storage Pools

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

Group Membership Restriction

Each VOB has a group list. If your principal *group* is on this list, you can create an element in that VOB. For more information about group lists, see the **protectvob** reference page in the *Command Reference*. Your principal *group is* the first group listed when you enter the **id(1)** command.

Importing Files

If you are adding a large number of files and directories to source control, use the **clearfsimport** command (or **clearexport** commands) and **clearimport** command. For more information, see the **clearfsimport** and **clearimport** reference pages in the *Command Reference*.

Moving, Removing, and Renaming Elements

This section explains how to move, remove, and rename elements.

Moving and Removing Elements

Because directories as well as files are under ClearCase control, you can move or remove elements from specific versions of directories without affecting the element itself. Moving or removing elements creates new versions of the parent directories to record the modifications.

For example, version 4 of /gui_vob/design contains an element named prog.c. If you remove prog.c from the design directory, ClearCase creates version 5 of /gui_vob/design, which does not contain the prog.c file element. The element prog.c itself is not modified.

```
cd pat_v1.4_cropcircle/gui_vob
cleartool ls design@@/main/4
  prog.c@@/main/2
  lib.c@@/main/10
cleartool checkout –nc design
   Checked out "design" version "/main/4"
cleartool rmname prog.c
  Removed "prog.c"
cleartool checkin –nc design
  Checked in "design" version "/main/5"
cleartool ls design@@/main/5
  lib.c@@/main/10
cleartool ls design@@/main/4
  prog.c@@/main/2
  lib.c@@/main/10
```

Before you move or remove an element name from a directory, verify with your project manager that your changes will not adversely affect other team members or break project builds.

To Move an Element Within a VOB

1. Check out the parent directory and the destination directory.
2. Enter the following command:

   cleartool mv *element-name destination-directory*
3. Check in the new parent directory and the source directory.

To Move an Element to Another VOB

Use the **cleartool relocate** command.

Warning: The **relocate** command makes irreversible changes to at least two VOBs and their event histories. We recommend that you not use it for minor adjustments. Furthermore, we recommend that you stop VOB update activity before and during a relocate operation. Check with your project manager and ClearCase administrator before using the **relocate** command.

To Remove an Element Name from a Directory

1. Check out the parent directory.
2. Enter the following command:

   cleartool rmname *element-name*
3. Check in the parent directory.

Other Methods for Removing Elements

Removing an element name from its parent directory does not affect the element itself, but two other types of a removal operation do irrevocably affect an element, and we recommend that you be very conservative in using these operations:

- Removing a version from the version tree of an element. For more information, see the **rmver** reference page in the *Command Reference*.
- Removing an element from a VOB. For more information, see the **rmelem** reference page in the *Command Reference*.

Renaming Elements

Renaming an element creates a new version of the parent directory to catalog the new element name. The element uses its new name in subsequent versions of its parent directory, but previous versions of the parent directory refer to the element by its previous name.

```
cd pat_v1.4_cropcircle/gui_vob
cleartool ls design@@/main/4
  prog.c@@/main/2
  lib.c@@/main/10
cleartool checkout –nc design
   Checked out "design" version "/main/4"
cleartool mv prog.c  msg.cat
  Moved "prog.c" to "msg.cat"
cleartool checkin design
  Default:
  Added file element "msg.cat".
Removed file element "prog.c".
Checkin comments for ".":  ("." to accept default)
.
  Checked in "design" version "/main/5"
cleartool ls design@@/main/5
  msg.cat@@/main/2
  lib.c@@/main/10
cleartool ls design@@/main/4
  prog.c@@/main/2
  lib.c@@/main/10
```

Before you move or remove an element name from a directory, verify with your project manager that your changes will not adversely affect other team members or break project builds.

To Rename an Element

1. Check out the parent directory.
2. Enter the following command:

cleartool mv *path target-path*

3 Check in the parent directory.

Accessing Elements Not Loaded into a Snapshot View

While working with source files in a snapshot view, you may need to see the contents of elements that are not loaded into the view or see ClearCase information about these nonloaded elements. For example, you may have chosen not to load a VOB that contains functional-specification documents. However, you may want to check periodically whether the functional specifications have been modified by reviewing the history of the elements.

Listing All Elements in the VOB Namespace

You can use the **cleartool ls** command to see all elements in the VOB namespace, even if they are not loaded into your snapshot view. This command lists the names of elements cataloged in the VOB namespace that the config spec of your view selects. The output of **cleartool ls** includes this information:

- The version ID of the particular version the view selects
- The version-selection rule in the config spec that selects this version

To see all elements in a directory, enter this command:

cleartool ls *path...*

For more information, see the **ls** reference page in the *Command Reference*.

Viewing the Contents of a Nonloaded Version

To access a version of a file not loaded into your view, use the **cleartool get** command, which copies the version you specify into your view. You can view nonloaded files or copy them into your view for build purposes, but you cannot check them out. Only file elements that are loaded into the view can be checked out.

Note: You cannot use **cleartool get** for directory elements.

To Copy a Nonloaded Version with cleartool get

To copy a nonloaded version of a file element into your view, type a command in this format:

cleartool get –to *filename version-extended-path*

For example:

cleartool get –to prog.c.previous.version prog.c@@/main/v3.1_fix/10

This command copies prog.c@@/main/v3.1_fix/10 into your view under the name of prog.c.previous.version. For information on version-extended paths, see *The Version-Extended Path* on page 58.

Registering a Snapshot View

If you need to perform ClearCase operations in a snapshot view that you did not create or have not updated, you must register that view.

When you create a snapshot view, ClearCase registers it in the file .ccase_svreg in your home directory. When ClearCase needs to access a snapshot view, it looks in that file in your home directory for the path for the snapshot view root directory. If you did not create or have not updated the view that ClearCase is trying to access, you must explicitly register the view.

If a snapshot view is not registered under your home directory and you use the snapshot view to perform ClearCase operations (for example, to check out a file), ClearCase cannot find the view. To use this view to perform ClearCase operations, register the view (see *To Register a Snapshot View* on page 87). If you inadvertently delete or corrupt the .ccase_svreg file, you must regenerate information in it for each snapshot view that you use (see *To Regenerate .ccase_svreg* on page 87).

To Register a Snapshot View

To register a snapshot view that you did not create or have not updated, perform the following steps:

1. Change your working directory to the root directory of the snapshot view.
2. From that directory, use either **cleartool update** or **cleartool update -print**.

This registers the view for your use on any UNIX workstation. The preview form of the command (**update -print**) is quicker than update and does not change the loaded files.

> **Note**: You cannot register a snapshot view that was created from a Windows host (see *Accessing Views Across Platforms of Different Types* on page 91).

For more information, see Chapter 4, *Updating a Snapshot View*.

To Regenerate .ccase_svreg

When you create a snapshot view or register a snapshot view, ClearCase creates or modifies the file .ccase_svreg in your home directory. Some ClearCase operations use information from this file. If you moved the snapshot view or the .ccase_svreg file in

your home directory has been deleted or corrupted, you must regenerate the file. To regenerate information in .ccase_svreg, follow the steps in *To Register a Snapshot View* on page 87 for each snapshot view that you use.

Moving Views

This section discusses the following tasks:

- Changing the physical location of the directory tree of a snapshot view
- Moving a view storage directory

For information about changing a view tag, see the **mktag** reference page in the *Command Reference*.

Changing the Physical Location of a Snapshot View

If the snapshot view storage directory is in a storage location, you can use the standard **mv** command to move the directory tree of loaded elements and view-private files of the snapshot view.

To Find the Location of the View Storage Directory

Enter the following command:

cleartool lsview –long *view-tag*

The Global Path field displays the path for the view storage directory.

> **Caution**: If the view storage directory is located below the root directory of the view (colocated with the view), **do not use** the standard **mv** command to move the snapshot view. Instead, see *Moving a View Storage Directory* on page 89.

If the snapshot view storage directory is in a storage location, you can move the view to a different workstation, but the workstation must run a UNIX operating system.

Update After Moving

After moving a snapshot view, you must use **cleartool update** (or **cleartool update –print**) to modify .ccase_svreg in your home directory (see *To Regenerate .ccase_svreg* on page 87). Some ClearCase operations use information from this file and will not succeed until you use **update** to modify it.

Moving a View Storage Directory

Each dynamic view and snapshot view includes a view storage directory, which ClearCase uses to maintain the view. **Do not use** the standard **mv** command to move a view storage directory for the following reasons:

- The view storage directory includes a database. Moving the database without first shutting down the *view_server* process for the view can corrupt the database.
- ClearCase stores the location of view storage directories in its own set of registries. The information in these registries must be correct for you to perform ClearCase operations in your views. In a dynamic view, the location in ClearCase registries must be correct for you to access any file or directory in the view.

We suggest that you ask your ClearCase administrator to move view storage directories because it may affect other, potentially many other, ClearCase users at your site. For more information about the procedure for moving view storage directories, see the *Administrator's Guide* for Rational ClearCase.

Caution: You will lose data (including view-private files in a dynamic view) if you move a view storage directory without following the procedure described in the *Administrator's Guide* for Rational ClearCase.

Regenerating a Snapshot View .view.dat File

The root directory of a snapshot view contains a hidden file, .view.dat. If you delete this file inadvertently, ClearCase no longer identifies the view as a ClearCase object, and you can no longer perform ClearCase operations on files or directories loaded in the view.

To Regenerate the .view.dat File

1. Open a command shell.
2. Type this command:

   Perl *ccase-home-dir***/etc/utils/regen_view_dot_dat.pl** ****
   [**–tag** *snapshot-view-tag* **]** *snapshot-view-path*

   For example:

   **Perl /usr/rational/etc/utils regen_view_dot_dat.pl **
   **–tag pat_v1.4_cropcircle_sv **
   ~/pat_v1.4_cropcircle_sv

   If the view storage directory is under the root directory of the view, you do not need to use the **–tag** *snapshot-view-tag* argument.

Accessing Views and VOBs Across Platform Types

ClearCase supports environments in which some ClearCase hosts use a Microsoft Windows operating system and others use a UNIX operating system.

This section discusses the following topics:

- Creating views across platform types
- Accessing VOBs across platform types
- Developing software across platform types

Creating Views Across Platforms of Different Types

Your ClearCase administrator can set up storage locations on Windows and UNIX server hosts. Any snapshot view that you create can use one of these storage locations, regardless of the platform type of the server host. For more information about storage locations, see the **mkstgloc** reference page in the *Command Reference*.

For a dynamic view, the view storage directory must be located on a host of the same platform type as the host from which you create the view. If you create a dynamic view from a UNIX host, you must locate the view storage directory on a ClearCase host on UNIX; if you create a dynamic view from a Windows host, you must locate the view storage directory on a Windows NT host that is set up to store view storage directories. We recommend that you locate dynamic view storage directories on the host from which you most often use the view.

Snapshot View Characteristics and Operating-System Type

For snapshot views, the operating system type from which you create the view determines view characteristics; the operating system type that hosts the files and processes related to a snapshot view do not affect the behavior of the view.

For example, it is possible to create a snapshot view from a Windows host and locate the view directory tree and the view storage directory on a ClearCase host on UNIX (assuming that you use third-party software to access UNIX file systems from Windows computers). Although all files related to the view are on a UNIX workstation, because you created the view from a Windows host, the view behaves as if its files are located on a Windows computer:

- Case-sensitive file lookups are not performed in the view.
- The view does not create symbolic links if the load rules encounter a VOB symbolic link.
- You can issue ClearCase commands for the view only from Windows hosts. (ClearCase hosts on UNIX do not recognize the directory tree as a snapshot view.)

Accessing Views Across Platforms of Different Types

This section describes support for accessing a view residing on a platform that differs from the platform from which it is being accessed.

Accessing UNIX Snapshot Views from Windows Hosts

ClearCase supports a set of third-party products for accessing UNIX file systems from Windows computers. If your organization uses one of these products, you can access UNIX snapshot views from Windows Explorer (or a command prompt) just as you would access any other directory tree on a UNIX workstation.

You can access snapshot views across platforms, but you cannot issue ClearCase commands across platforms. For example, you cannot check out files in UNIX snapshot views from Windows hosts nor can you create shortcuts to UNIX snapshot views from ClearCase Explorer.

If, from a Windows host, you hijack a file in a UNIX snapshot view, ClearCase detects the hijack when you update the view from a ClearCase host on UNIX.

Accessing Windows Snapshot Views from UNIX Hosts

ClearCase does not support accessing Windows file systems from UNIX workstations.

Accessing UNIX Dynamic Views from Windows Hosts

ClearCase supports a set of third-party products for accessing UNIX file systems from Windows computers. If your organization uses one of these products, you can complete the following tasks to access UNIX dynamic views from Windows computers:

1. Create the UNIX view with the proper text mode. For more information, see *Developing Software Across Platforms of Different Types* on page 92.
2. Import the view tag of the UNIX view into your Windows network region.
3. Start the dynamic view.

Accessing Windows Dynamic Views from UNIX Hosts

ClearCase does not support products for accessing Windows file systems from UNIX workstations. You cannot access Windows views from UNIX hosts.

Accessing VOBs Across Platforms of Different Types

Your ClearCase administrator sets up VOBs on Windows or UNIX hosts and creates *VOB tags* in each ClearCase network region that needs to access the VOBs. (For information about registering UNIX VOB tags in a Windows network region, see the *Administrator's Guide* for Rational ClearCase.) Then, from any ClearCase host on Windows or UNIX systems, you can create snapshot views to load elements from VOBs that have tags in your network region.

From a ClearCase host on Windows that supports dynamic views, you can access VOBs on Windows and UNIX from dynamic views as well as snapshot views. To access VOBs on UNIX from Windows dynamic views, you must use third-party software that provides access to UNIX file systems from Windows computers. From a ClearCase host on UNIX, you cannot access VOBs on Windows from dynamic views. Table 2 summarizes your options for accessing VOBs across platform types.

Table 2 Accessing ClearCase VOBs Across Platform Types

Platform of your ClearCase host	Platform on which VOB is located	View from which you can access source files
Windows computer	Windows computer or UNIX workstation	Snapshot view or dynamic view
UNIX workstation	Windows computer	Snapshot view
UNIX workstation	UNIX workstation	Snapshot view or dynamic view

Developing Software Across Platforms of Different Types

If developers check in source files from views created on both Windows and UNIX hosts, consider creating your views in interop (MS-DOS) text mode. The text modes change how a view manages line terminator sequences. For more information about view text modes, see the *Administrator's Guide* for Rational ClearCase or ClearCase Help. For information about starting Help, see *To Access Online Information* on page 1.

Working in a Snapshot View Without the Network

A

If you need to work with your source files from a computer that is disconnected from the network of Rational ClearCase hosts and servers, you can set up a snapshot view for disconnected use.

This chapter describes the following tasks:

- Setting up a view for your hardware configuration
- Preparing the view
- Disconnecting the view
- Working in the view
- Reconnecting to the network
- Using the Update Tool

Note: While disconnected from the network, you cannot access ClearCase information about the files in your view or issue most ClearCase commands. If you want to work from a remote location and continue to access ClearCase information and issue ClearCase commands, consider using the Web interface for ClearCase. Ask your ClearCase administrator whether the Web interface for ClearCase has been configured at your site and what URL you need to supply to your Web browser to access it. For information about using the Web interface, see the Web interface Help.

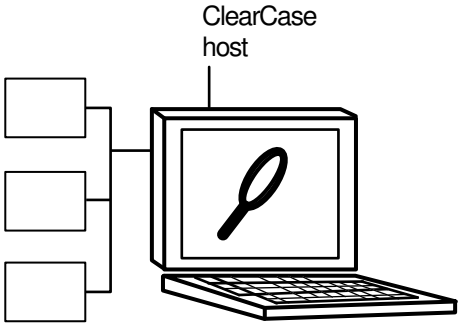
Setting Up a View for Your Hardware Configuration

You can use one of several hardware configurations to work in a snapshot view that is disconnected from the network.

This chapter describes the following recommended configurations:

- Creating and using the view on a laptop computer that periodically connects to the network (Figure 21).

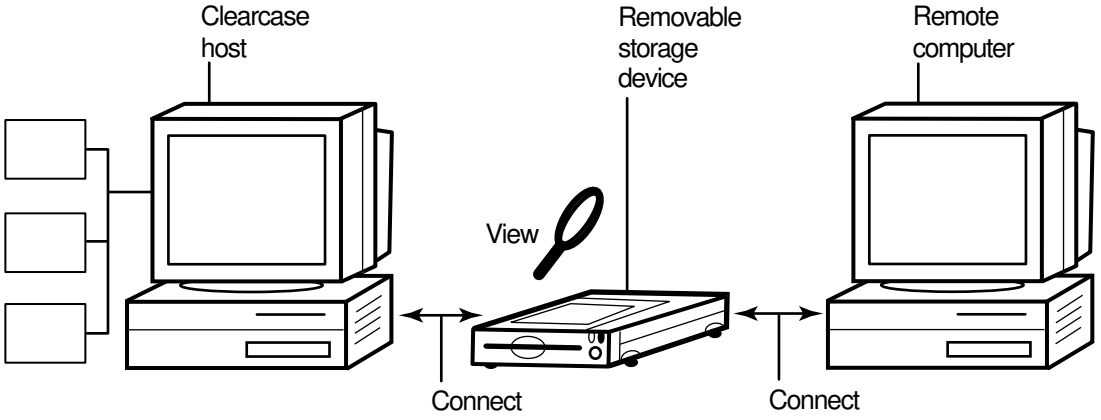
Figure 21 View on a Laptop



Note: The laptop computer must run a UNIX operating system.

- Creating and using the view on a removable storage device such as an external hard drive or some other device (such as a Jaz drive) that provides satisfactory read/write performance (Figure 22).

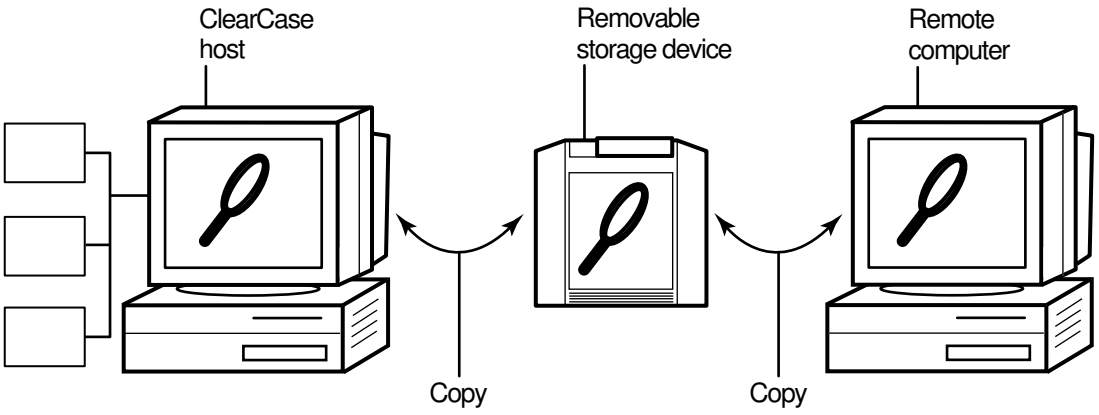
Figure 22 View On a Removable Storage Device



Note: The remote computer must run a UNIX operating system.

- Copying the view from a ClearCase host to a temporary, removable storage device such as a diskette or a tape drive, which usually does not provide satisfactory read/write performance, and then copying the view from the storage device to a computer that is disconnected from the network (Figure 23).

Figure 23 Copy the View



Note: The remote computer must run a UNIX operating system.

Under the Hood: View Storage in Disconnected-Use Configurations

In all the configurations recommended for disconnected use, the snapshot view storage directory is in a server storage location. We recommend this configuration because the *view_server* process of a view runs on the host that contains the view storage directory or on a host that can access a location on a supported Network Attached Storage (NAS) device which contains the view storage directory.

A view_server is a long-lived process that manages activity for a specific view. If the view storage directory is in the root directory of the snapshot view and you disconnect the view from the network while the view_server process is writing to the storage directory, you can corrupt the data ClearCase uses to maintain your view.

Preparing the View

Before you disconnect the view from the network, complete these tasks:

- Update the view to establish a checkpoint. (For information about updating the view, see *Updating the View* on page 101.)
- Check out the files you expect to modify. After you are disconnected from the network, you cannot check out files, although there are workarounds. (See *Hijacking a File* on page 96.)

When you are no longer connected to the network, you cannot use most ClearCase commands. At this point, the disconnected computer does not distinguish a snapshot view directory from any other directory in the file system.

Disconnecting the View

If the view is located on a laptop or removable storage device, disconnect the device from the network; reconnect the removable media to a remote computer.

If you do not have a storage device with satisfactory read/write performance, use a standard UNIX copy command to copy files from your view to the storage media and from the storage media to the remote computer. To prevent ClearCase from identifying copied files as *hijacked*, use copy command options to preserve file times. For example:

```
cp -Rp
```

Working in the View

You cannot use most ClearCase commands when disconnected from the network. Yet you may need to work on files that you did not check out or locate files you have modified. This section provides workarounds for these ClearCase operations.

Hijacking a File

If you need to modify a loaded file element that you have not checked out, you can *hijack* the file. ClearCase considers a file hijacked when you modify it without checking it out. For more information, see *Under the Hood: Determining Whether a File Is Hijacked* on page 100.

When you reconnect to the network, you use the Update Tool to find the files you hijacked. You can do the following with a hijacked file:

- Check out the file. You can then continue to modify it and, when you are ready, check in your changes.
- Undo the hijack. For more information, see *To Undo a Hijack* on page 100.

To Hijack a File

Use **chmod** to add the **write** permission and then modify the file. For example:

chmod +w prog.c

To Find Modified Files While Disconnected

To find all files that have been modified within a specified number of days, use the following command:

find *snapshot-view-path* **–mtime –***number-of-days* **–ls –type f**

For example, to find all files modified within the last two days, enter this command:

find ~/pat_v1.4_cropcircle –mtime -2 –ls –type f

For more information, see the **find** manpage.

Reconnecting to the Network

If the view is located on a laptop or removable storage device, connect the device to the LAN and make sure that the view is accessible to the host on which the view storage directory is located.

If you copied the view onto removable media, use a standard UNIX copy command to copy files back to the original location on the network computer.

Using the Update Tool

When you are connected to the network, use the Update Tool for the following tasks:

- Determine how to handle hijacked files
- Update the view

Determining How to Handle Hijacked Files

Handling hijacked files involves the following tasks:

- Finding hijacked files
- Comparing a hijacked file to the version in the VOB

- Checking out a hijacked file
- Merging changes to a hijacked file
- Undoing a hijack
- Choosing other ways to handle hijacked files

To Find Hijacked Files

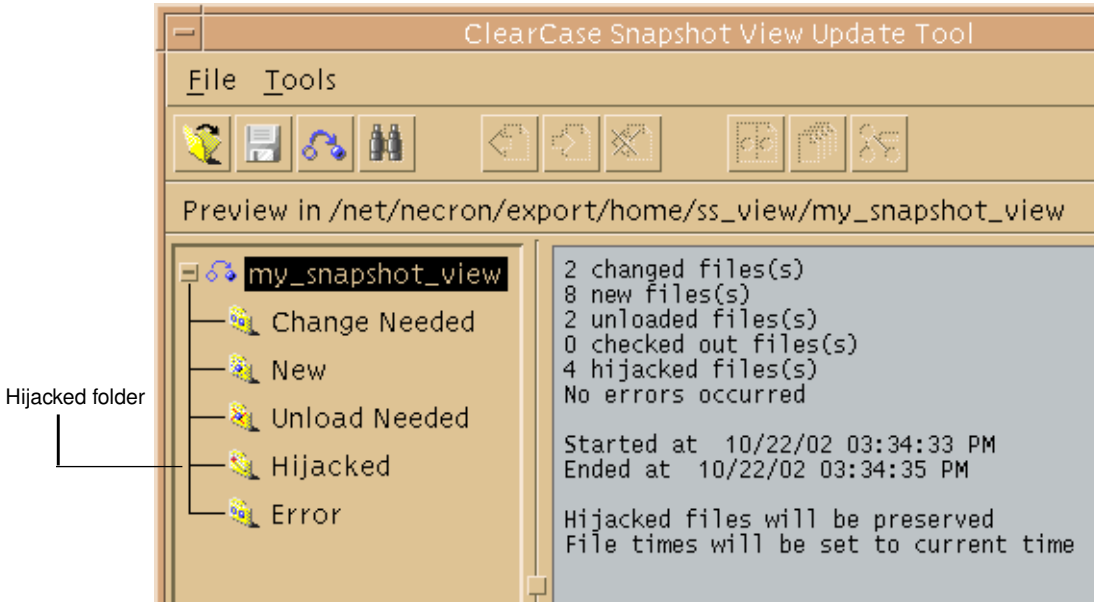
1 Enter the following command:

cleartool update –graphical *snapshot-view-path*

2 In the Update dialog box, click **Preview only**. Then click **OK**.

3 If any hijacked files are in your view, the Snapshot View Update window displays a folder in the left pane titled **Hijacked** (Figure 24). Select **No** for the option asking whether you want to check out the hijacked files now.

Figure 24 Hijacked Files in the Update Window



To Compare a Hijacked File to the Version in the VOB

You can use the Diff Merge tool to see how the hijacked file differs from the checked-in version of the file:

1 In the right pane of the Snapshot View Update window, click a hijacked file.

2 Click **Tools > Compare with old**. For information about using the Diff Merge tool, see the Help.

To Check Out a Hijacked File

To keep the modifications in a hijacked file, check out the file:

1 In the right pane of the Snapshot View Update window, click a hijacked file.

2 Click **Tools > Checkout**.

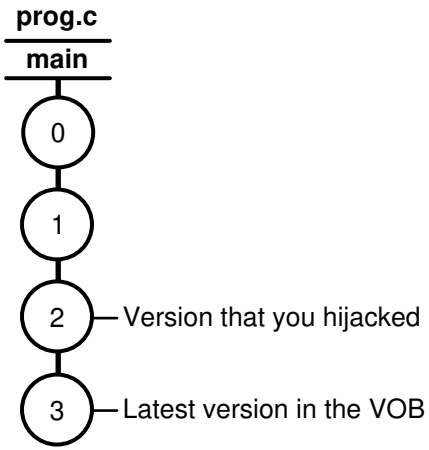
3 ClearCase treats a checked-out hijacked file as it does any other checkout.

When you are ready, you can check in the file.

Merging the Latest Version to a Hijacked File

If you are working with a shared set of versions and someone has checked in a newer version of the file while it was hijacked in your view (Figure 25), ClearCase prevents you from checking in the file.

Figure 25 Hijacked Version May Not Be the Latest Version



You have to merge the latest version in the VOB with the checked-out file before ClearCase allows the checkin.

To merge the latest version in the VOB to the checked-out version in your view, enter the following command:

cleartool merge –graphical –to *file-or-directory-in-your-view* \
*file-or-directory-name***@@/main/LATEST**

Note: @@/main/LATEST is a *version-extended path*. For more information, see *The Version-Extended Path* on page 58.

For example:

```
% cleartool merge –graphical –to prog.c   prog.c@@/main/LATEST
```

Using the **–graphical** option starts the Diff Merge tool. For information about using the Diff Merge tool, see the Help for ClearCase. After merging, save the results and check in the version by entering the **cleartool checkin** command from the view.

To Undo a Hijack

If, for specific hijacked files, you want to discard your changes and get a fresh copy of the version from the VOB, you can undo the hijack.

1. In the right pane of the Snapshot View Update window, select one or more hijacked files.
2. Click the selected files, and click **Tools > Undo hijacked file**.

ClearCase overwrites the hijacked file with the version that was loaded in the view. If you want to overwrite hijacked files with the versions the config spec selects in the VOB, see Step 2 in *Updating the View* on page 101.

Under the Hood: Determining Whether a File Is Hijacked

To keep track of file modifications in a snapshot view, ClearCase stores the size and last-modified time stamp of a loaded file (as reported by the UNIX file system). ClearCase updates these values each time you check out a file, check in a file, or load a new version into the view.

To determine whether a file is hijacked, ClearCase compares the current size and last-modified time stamp of a non-checked-out file with the size and time stamp recorded in the view database. If either value is different from the value in the view database, ClearCase considers the file hijacked.

Changing the read-only permission of a non-checked-out file does not necessarily mean ClearCase considers the file hijacked.

Other Ways to Handle Hijacked Files

While updating the view, you can handle hijacked files in any of the following ways:

- Leave hijacked files in place
- Rename the hijacked files and load the version from the VOB
- Overwrite hijacked files with the version the config selects in the VOB

For more information, see *Updating the View*.

Updating the View

1. Enter the following command:

   cleartool update –graphical *snapshot-view-path*

2. To configure the Update Tool for handling hijacked files, in the Update dialog box click the **Advanced** tab and select a method for handling the remaining hijacked files. You have these choices:

   - Leave hijacked files in place
   - Rename the hijacked files and load the selected version from the VOB
   - Delete hijacked files and load the selected version from the VOB

3. To start the update, click **OK**.

Index

.ccase_svreg file 87
.cqparams file 35
.keep files, canceled checkouts 40
.unloaded files, how created 52
.view.dat file
  about 14
@@ extended naming symbol 82
@@ notation 58

A

access mode 79
adding files to source control 77–78
auto-make-branch 81

B

backslash (\), continuing command lines 17
branches
  about 56
  how used 58
  mastership issues in MultiSite 73
  mastership request procedures 75
  merging and mastership 75
  merging parts of subbranches 68
  merging subbranches 67
  merging, tools for 60
build auditing and build avoidance 40
building software
  ClearCase build tools 40

C

checking in
  about 42
  compared to update operation 49
  effect of on VOB links 23
  how it works 45
  merging latest with checked-out version 43
  procedure for 43
checking out
  before transfer of mastership 75
  for remote use 96
  hijacked files 99
  how handled 33
  nonlatest version 43
  nonloaded files 86
  procedure 30
  when disconnected from network 96
checkouts
  about 31
  how cancellation is handled 41
clearexport and clearimport commands 83
ClearQuest integration
  about 6
  checking in 45
  checking out with 34
  command line interface 36
  graphic user interface 35
ClearQuest, setting up user database defaults 24
cleartool
  about 17
comparing versions
  hijacked files 98
  procedures 38

config specs
about 3
adding or changing load rules 20
creating 6
for snapshot views 3
role in snapshot view checkouts 33
role in update operation 51
use in branches 58
copying
nonloaded versions into views 86
snapshot views to removable storage devices 96
views from removable storage devices 97

D

derived objects
nonshareable conversion 80
describe command, -cview option 39
development policy
on checkouts 31
on snapshot view location 12
development tasks
use of branches 58
directories
adding to source control 77
canceling checkouts 41
comparing and merging versions 60
finding checkouts from 40
importing directory trees to source control 83
listing nonloaded files 86
merge algorithm 66
merge procedure 66
remote use 94
removing element names 84
unloading 53
disk space requirements
snapshot views 12
documentation, accessing online 1
dynamic views
accessing across network 30
behavior of VOB links 22
build tools 40
creating 17
handling checkouts 34
starting work in 30
view storage directory location 14
when to use 12

E

element types 79
elements
about 2
access mode 79
auto-make-branch during creation 81
conversion of nonshareable DOs to 80
creation in replicated VOBs 81
directory creation 81
history of changes 38
moving and removing 83
nonloaded, accessing 86
object and version references 82
renaming 85
selecting for view 3
unloading from snapshot views 52
updating in snapshot views 51
extended naming symbol (@@) 82

F

file attributes
removing Read-Only 33
file types 79
files
accessing 29
adding to existing directory tree 77
adding to source control 77
checking out 30
finding checked-out 40
listing nonloaded 86
unloading from snapshot view 52
VOB link 23

G

get command 86
group membership 83

H

hard links 22
hardware configurations for remote use 94
help, starting online 1
hidden file 89
hijacked files
about 96
checking out 99
comparing to version on VOB 98
finding 97–98
handling 97
how determined 100
merging 99
undoing hijack 100
unloading from snapshot view 52
History Browser 38

I

importing directories to source control 83
integration
base ClearCase-ClearQuest 6
interoperation on Windows and UNIX 90–92

L

laptops
configuration for remote use 94
line continuation character (\) 17
load rules
adding or changing 20
excluding elements 21
illustration 3
ls command 86

M

Merge Manager 60
merging directories
algorithm 66
procedure 66
merging files
all changes 67
at checkin 43
branch mastership in MultiSite 73
directory versions 60
hijacked files 99
how it works 61
non-ClearCase tools 72

removing changes (subtractive merge) 71
selective versions 68
tools for 60
mkview command 16
MultiSite, sharing branch in 73
MVFS 4

N

namespace 8
resolving differences between views 39

O

online information 1

P

parallel development 8, 55
paths
invalid, resolving 39
version-extended 58

R

read/write performance of remote storage devices 94
relocate command 84
reserved checkouts 31

S

shortcut menus, deactivated 89
snapshot views
See also updating snapshot views
access to nonloaded elements 86
adding or changing load rules 19
behavior of and operating system 90
behavior of symbolic links 22
choosing locations for 12
config specs 3
copying nonloaded versions to 86
copying to removable storage devices 96
creating 16
excluding elements 21
handling checkouts 33
hardware configurations for remote use 94
loading 29
location of storage directory 13
moving 88
registering 87
tool for 97
transferring to laptop 96
view.dat file 89
when to use 11
startview command 39
storage devices, removable
disconnecting from network 96
performance of views copied to 94
storage pools 83
symbolic links 22

T

tutorials 1

U

umask
  setting at view creation time 15
under the hood
  .ccase_svreg file 17
  adding files to source control 78
  canceling checkouts 41
  checking in files 45
  checking out files 33
  hijacked files, how determined 100
  how merging works 61
  initial version on a branch 56
  updating snapshot views 51
unloading files and directories
  causes in update operation 52
unreserved checkouts 31
Update Tool
  about 97
  detecting hijacked files 96
  handling hijacked files 101
updating snapshot views
  canceled directory checkout 42
  compared to checkin 49
  files and directory trees 51
  handling hijacked files 101
  how it works 51
  moving the view 88
  procedure 49
  remote use of view 96
  scope 49
  unloading elements 52
  VOB links 23

V

version IDs, viewing 86
version trees
  about 55
version-extended paths 58
  labels in 5
versions
  about 2
  copying nonloaded into views 86
  initial on branches 56
  merging all changes to one element 67
  merging directories 60
  merging outside ClearCase 72
  merging specific on branch 68
  removing merged changes 71
  reserving right to create 31
version-selection rules
  about 4
  adding or modifying 18
  illustration 3
  listing for elements 86
  on branches 58
  unloaded elements 52
  update operations 51
view storage directories
  about 12
  disk space required 12
  location for dynamic views 14
  location for remote use 95
  moving 89
view tags
  about 13
view.dat file
  regenerating 89
view-private files 6
views

about 2
creating on Windows and UNIX 90
moving 88
naming 13
types of 2
VOB namespace
about 19
listing elements in 86
VOBs
about 2
activating for dynamic view 30
using multiple 3

W

working from a remote location
about 93
hardware configurations 94
removable storage devices 94
updating view 96