Working in Base ClearCase

Contents

Working in Base ClearCase

Figures

1 ClearCase Concepts

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

- ClearCase views
- Versions, elements, and VOBs
- Parallel development

1.1 Recommended Reading Paths

Read this chapter first. Then, if you want to start working immediately, use online 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 your organization's development cycle. To start online help, type this command:

```
hyperhelp cc_main.hlp
```

The sections titled *Under the Hood* provide detailed information and suggest ways to become an advanced ClearCase user.

1.2 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

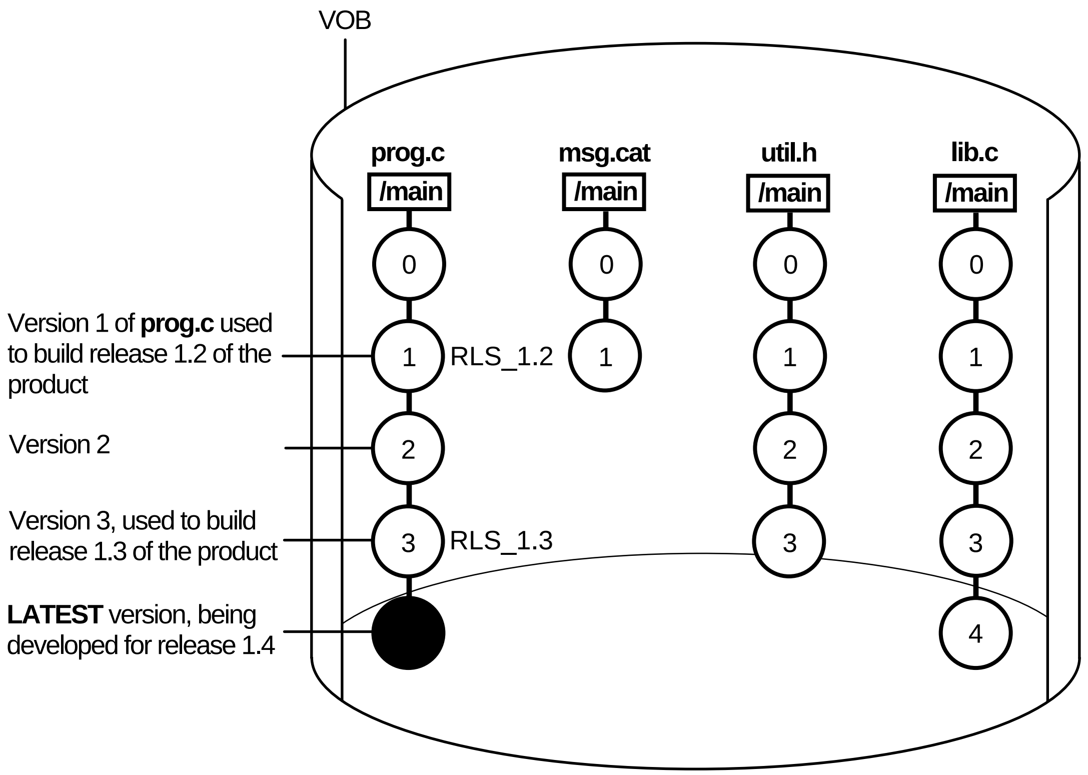
ClearCase includes two kinds of views:

- ➤ *Snapshot views*, which copy files from data repositories called *VOBs* (*versioned object bases*) to your computer.
- ➤ *Dynamic views*, which use the ClearCase *multiversion file system* (MVFS) to provide immediate, transparent access to the data in VOBs. (Dynamic views may not be available on all platforms. For more information, see ClearCase online help.)

1.3 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


VOB
prog.c
/main
msg.cat
/main
util.h
/main
lib.c
/main
0
0
0
0
Version 1 of **prog.c** used to build release 1.2 of the product
1
RLS_1.2
1
1
1
Version 2
2
2
2
Version 3, used to build release 1.3 of the product
3
RLS_1.3
3
3
LATEST version, being developed for release 1.4
4

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.

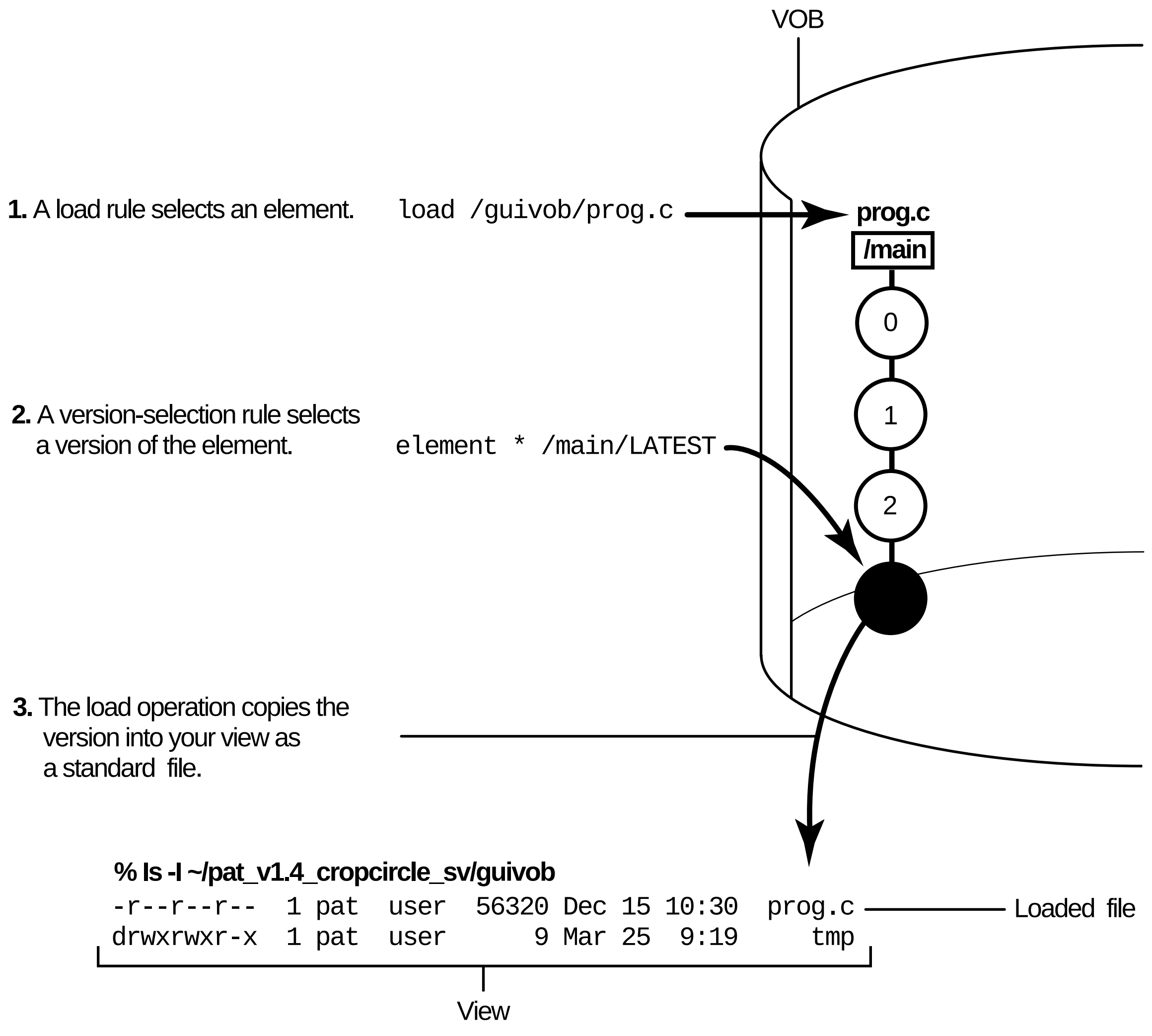
Selecting Elements and Versions

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**.

Figure 2 Config Spec for Snapshot Views


VOB
1. A load rule selects an element.
load /guivob/prog.c
prog.c
/main
0
1
2
2. A version-selection rule selects a version of the element.
element * /main/LATEST
3. The load operation copies the version into your view as a standard file.
Loaded file
View

```
% ls -l ~/pat_v1.4_cropcircle_sv/guivob
-r--r--r--  1 pat  user  56320 Dec 15 10:30  prog.c
drwxrwxr-x  1 pat  user      9 Mar 25  9:19     tmp
```

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*.
- 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 Pathnames

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 pathname. 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 pathnames are equivalent:

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

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

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

This version-extended pathname 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. These non-ClearCase file system objects are called *view-private objects.*

1.4 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're working on release 1.4 of a software product, and you want to experiment with the GUI 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're 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.

1.5 Extended Namespace for Elements, Branches, and Versions

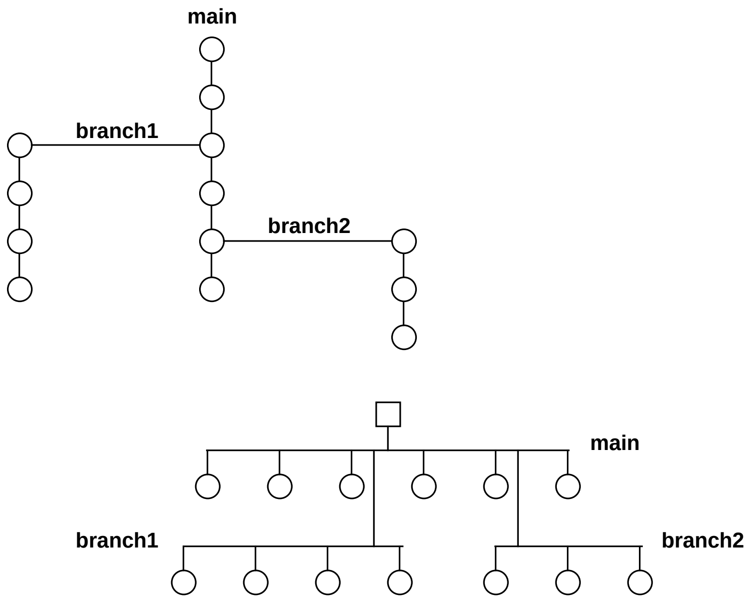
An element’s version tree has the same form as a standard directory tree (Figure 3), which compares components of the version tree to components of a directory tree in extended namespace.

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.)

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.

Figure 3 Version Tree and Extended Namespace



Accordingly, any location within an element’s version tree can be identified by a pathname in this extended namespace:

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

1.6 The Base ClearCase-ClearQuest Integration

Rational ClearQuest is a change-request management system that can integrate with base ClearCase projects to provide extended features. These features enable you to associate change requests in a ClearQuest user database and versions of ClearCase elements on which you are working.

The Base ClearCase-ClearQuest Schema and User Databases

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

The ClearQuest schema package provides additional information in your ClearQuest records to track associations with ClearCase versioned objects. To support associations between base ClearCase and ClearQuest record types, the schema repository needs to have the schema package applied to its record types and the user database used with the integration 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 database supplied with the base ClearCase-ClearQuest integration is called a defect, but with the ClearCase schema package installed, any record type can be used.

ClearCase Triggers and ClearQuest 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 ClearQuest change requests with one or more ClearCase 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 integration provides a text-based user interface for users of the **cleartool** command-line interface and a **clearprompt** pop-up window interface for users of the ClearCase GUIs such as ClearCase Explorer and Windows Explorer (on Windows computers) and ClearCase File Browser (on UNIX workstations).

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

- Associate a version with one or more change requests when you check out or check in the element.
- 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.

As a ClearQuest user, you can do the following:

- View the change set for a change request.
- See the files 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 that determines the conditions under which you are prompted to associate versions with change requests.

- 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 ClearQuest administrator adds the ClearCase schema package to a ClearQuest schema. The administrator sets a policy for one or more VOBs that specifies the conditions under which you are prompted to associate versions with change requests.

2 Setting Up a View

Usually you set up 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

2.1 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 ClearCase online help.)

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 doesn’t use the ClearCase *build auditing* and *build avoidance* features.

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.

2.2 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’s 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's **view_server** process 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 view's owner or contents; 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

A view's name 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-pathname

  You do not refer to the view-tag when performing most ClearCase operations. Instead you usually refer to the view's pathname.

  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's **.view.dat** file, it can determine the view's view-tag. (If you delete the **.view.dat** file inadvertently, see *Regenerating a Snapshot View's .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 view's directory tree. For more information, see *Accessing Files* on page 27 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. View processes (specifically, **view_server** processes) run on the computer that physically stores 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.

2.3 Adjusting Your umask

Your **umask(1)** setting at the time you create a view affects how accessible it will be 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, refer to a **umask(1)** man page.

The CCASE_BLD_UMASK Environment Variable

You can also use the CCASE_BLD_UMASK environment variable (EV) 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 on ClearCase EVs, see the **env_ccase** reference page in the *Command Reference*.

2.4 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 *pathname-for-view*

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

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 *Regenerating .ccase_svreg* on page 90.

To Create a Dynamic View

Enter the following command:

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

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 main 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*.

2.5 Adding or Modifying Version-Selection Rules

ClearCase creates a set of default version-selection rules in your view's initial config spec. 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 on creating version-selection rules, see *Managing Software Projects*.

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

To Copy or Include Version-Selection Rules

1. Do one of the following:

- If you're working in a dynamic view, type the following command:

  cleartool setview *view-tag*

- If you're working in a snapshot view, change to the root directory of the view.

2. Type the following command:

cleartool edcs

ClearCase opens the view's config spec 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 project's rules 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're working in a snapshot view, create or modify *load rules*. (See *Snapshot View: Adding or Modifying Load Rules* for more information.) If you're working 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 ClearCase prompt for setting the config spec.

2.6 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 your view’s config spec 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. From a snapshot view, 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 *VOB-pathname* [...]

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

```
cleartool ls /guivob
drwxrwxrw-  1 pat   user  9   Dec 15 10:30  batch
drwxrwxrw-  1 pat   user  3   Dec  8  8:10  soup
cleartool ls /guivob/batch
-r--r--r--  1 pat   user  56320 Mar  25  9:15  prog.c
-r--r--r--  1 pat   user  45056 Mar  23 11:57  lib.c
```

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 a snapshot view's config spec, ClearCase updates the entire view. This is appropriate when you first create a view, or when you modify a view's version-selection rules, 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 your view's config spec 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 view's config spec 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 view's config spec in your default text editor.

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

   load *vob-tag/element-pathname* [...]

   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 ClearCase prompt for setting the config spec.

To Add Load Rules with update –add_loadrules

Enter the following command:

cleartool update –add_loadrules *element-pathname* [...]

element-pname names an element in the VOB namespace at a pathname 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
```

You can also use a relative pathname for the *element-pathname* 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 an element's initial version. For all ClearCase elements, the initial version contains no data.

To Exclude Elements

1. Open the view's config spec 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-pathname* **/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 ClearCase 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 alternate 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 pathname, ClearCase considers other pathnames for the element as "checked out but removed." That is, to prevent you from modifying the element from multiple pathnames, ClearCase executes standard checkout behavior at only one pathname (the one from which you entered the **checkout** command), but does not create view-private files at other pathnames. 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.

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.

3 Working in a View

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

3.1 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, if you use this command to create the view

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

the view's files 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 directory's owner 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's view storage directory. For more information, see the *Administrator's Guide* for Rational ClearCase.

3.2 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 30.

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

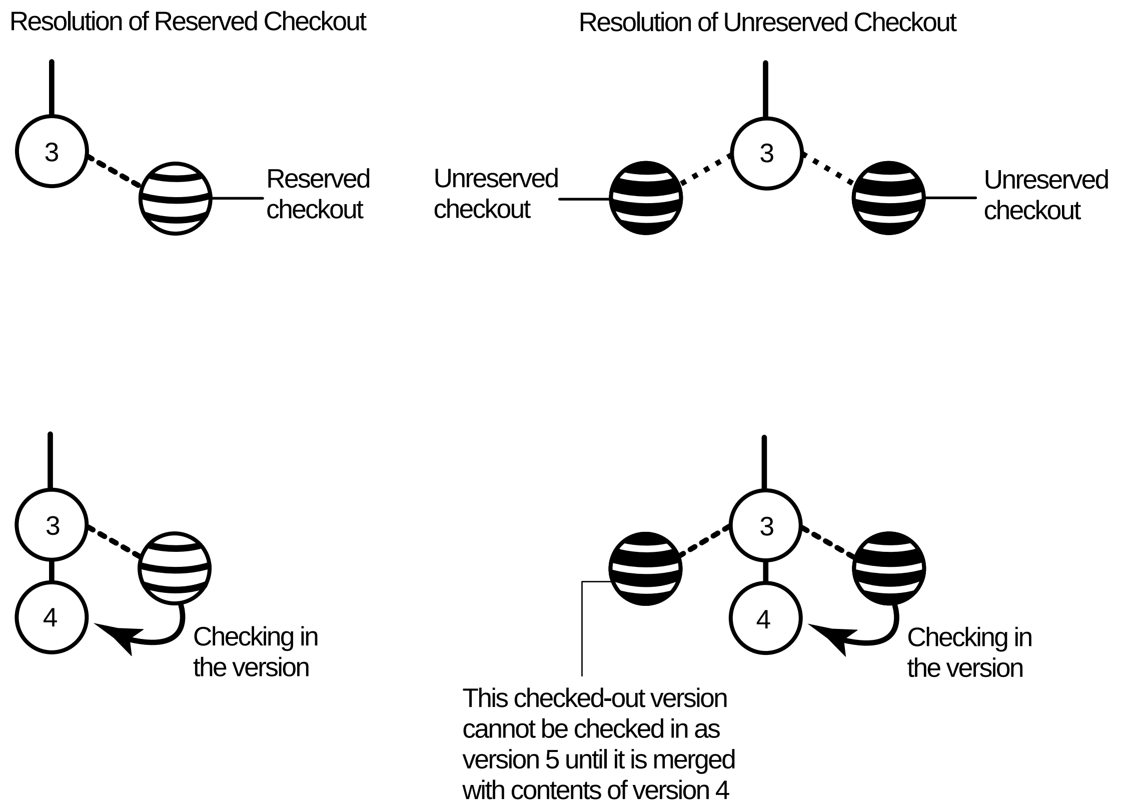
Reserved and Unreserved Checkouts

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. Your organization's development policy 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.

Figure 4 Resolution of Reserved and Unreserved Checkouts


Resolution of Reserved Checkout
Resolution of Unreserved Checkout
3
Reserved checkout
Unreserved checkout
3
Unreserved checkout
3
4
Checking in the version
3
4
Checking in the version
This checked-out version cannot be checked in as version 5 until it is merged with contents of version 4

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 71.

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 on 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*.

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.

From a Snapshot View

When you check out a file or directory from a snapshot view, the ClearCase software handles the request 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.

   ➢ Your view's config spec selects versions based on a label or a time rule, and the latest version in the VOB falls outside those parameters (Figure 5).

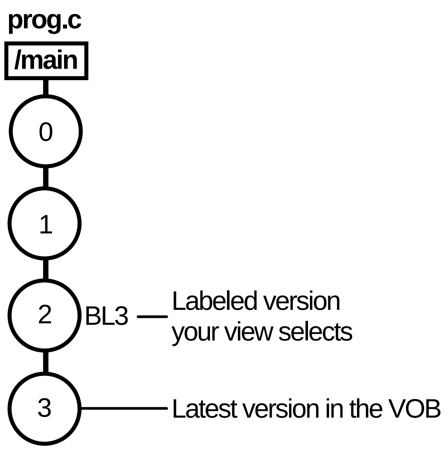
3. If you check out a version other than the one currently loaded in your view, ClearCase loads the checked-out version into your view.

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 on checking out VOB links in a snapshot view, see *Under the Hood: VOB Links* on page 23.

Figure 5 Selecting the Non-Latest Version of an Element



From a Dynamic View

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

1. If your view's version-selection rules 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.

   See *Merging with the Latest Version* on page 43 for information about checking in a nonlatest version.

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 change request record. For more information, see *The Base ClearCase-ClearQuest Integration* on page 8.

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 Base ClearCase-ClearQuest Integration 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 Windows computers:

```
<windows>\Profiles\<login>\Application Data\Rational\Cqcc\
```

On UNIX workstations, it 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 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.

Using the Base ClearCase-ClearQuest Integration Interface

If you are logged in to a ClearQuest user database (see *Logging on to a ClearQuest User Database* on page 34) and check out or check in an element, you see a different interface. You have the options shown in Table 1.

Table 1 Base ClearCase-ClearQuest Integration Options

Option	Description
OK - commit associations	Make the requested associations and exit.
CANCEL	Exit without making any changes and cancel the related ClearCase operation.
HELP	Print 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 local administrator defines to show defects that are appropriate. Depending on your local policy, you select one or multiple items.
TYPEIN - Enter Associations from Keyboard	Allows you to enter one or more change-request IDs directly.
DATABASE	Shows the current database name; allows you to change to a different database.
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 database.
PATHNAME	Shows the full path for the version that you are checking in or checking out. Select this item to see more information.

If you use a **cleartool** command line interface (CLI), you see a numbered menu of the options shown in Table 1. To select an option, type the number of the entry and press RETURN. Some menus allow multiple choices, to which you can enter one or more number separated by a space or a comma. To dismiss the menu without making a choice, simply press RETURN.

If you use File Browser, you see the **ClearQuest Associations Menu** provided by ClearPrompt. The dialog box opens when you check out or check in a version of an element in a VOB enabled for ClearQuest. In the dialog box, select an option from those shown in Table 1 and click **Yes** to continue. If multiple choices are allowed, click multiple items. Clicking the same choice a second time clears it as a choice. To close the dialog box without making a change, click **Yes** or **Abort**.

Both the CLI and GUI interfaces use the same text for the choices to associate a ClearCase version with one or more ClearQuest change requests and to remove a version's existing associations with change requests.

Associating a Checkout with a ClearQuest Entity

If you 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 your specified and complete the checkout operation.

3.3 Working with Checkouts

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

➤ Viewing an element's history
➤ 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 an Element's History

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

To View an Element's History

In a view, enter this command:

cleartool lshistory –graphical *pathname*

You can use this command from a snapshot view whether or not the element specified by *pathname* 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 *pathname*

To Compare with a Version Other Than the Predecessor

1. In a shell, enter this command:

   cleartool lsvtree –graphical *pathname*

2. In the Version Tree Browser, select a version.

3. Click **Version** > **Diff** > **Selected vs. Other**.

4. In the **Enter other versions** dialog box, select other versions and click **OK**.

If you prefer to use the command line:

1. Use **cleartool lsvtree** to list an element's versions.

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 *pathname* is loaded into the view. For more information, see the **diff** and **pathnames_ccase** reference pages in the *Command Reference*.

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* pathnames in different views. ClearCase can track directory-level changes, from simple renamings to wholesale reorganizations. In such situations, a team member may direct your attention to a particular element, using a pathname that is not valid in your view. You can pass this pathname to a **describe –cview** command to determine its pathname in your own view. For example:

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

```
file element "/guivob/lib.c@@"
  created 20-May-93.14:46:00 by 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 /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're 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*.

3.4 Canceling Checkouts

If you check out a file but don't 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 *pathname*

   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 config spec's version-selection rules 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 pathname of the new element) to the VOB's **lost+found** directory 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 on starting the Update Tool, see ClearCase online help.

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 orphaned element's name and use this command:

cleartool rmelem *VOB-pathname***/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 element's parent directory 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.

3.5 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 element's history. We recommend that you check in a file or directory any time you want a record of its current state.

Ideally, your organization's development strategy 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

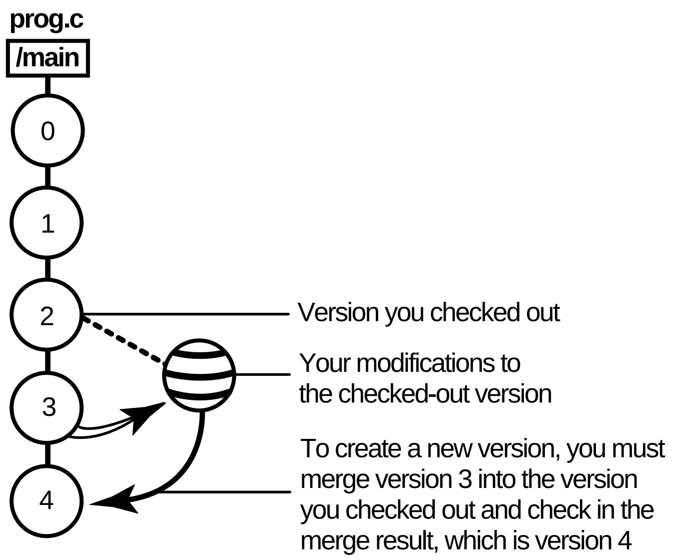
Prompts for and appends a single additional comment to all elements in the list.

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

Merging with the Latest Version

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).

Figure 6 Merging with the Latest Version


prog.c
/main
0
1
2
3
4
Version you checked out
Your modifications to the checked-out version
To create a new version, you must merge version 3 into the version you checked out and check in the merge result, which is version 4

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. (The section *Under the Hood: What Happens When You Check Out a File or Directory* on page 32 describes the situations in which you may have to merge before checking in.) Note that the reserve status of the checkout is not relevant to whether your modifications can be checked in.

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 with 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 pathname*. For more information, refer to the **pathnames_ccase** reference page in the *Command Reference*.

For example:

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

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

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.

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 view’s config spec.

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 your view’s directory tree.)

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 config spec's version-selection rules 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. Using the options in Table 1, you can do the following:

➤ Keep the same change-request IDs.

  Use the QUERY option to see a list of all open change-request IDs currently assigned to you.

➤ Delete some or all of the change-request IDs.

  Use the REVIEW option to list and possibly delete current associations.

➤ Add new change-request IDs.

  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.

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.

View the Versions for a Change Request from ClearQuest

To view the ClearCase 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.

4 Updating a Snapshot View

The rules in your view's *config spec* 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.

4.1 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 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-pathname*

The *snapshot-view-pathname* 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. **–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 view's state 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. **–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 to update the view instead of the command line. To start the update GUI, enter the following command:

cleartool update –graphical *snapshot-view-pathname*

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] *pathnames-of-loaded-elements*

For information on these command options, see *Updating the View* on page 48. For a description of 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 39.

4.2 Under the Hood: What Happens When You Update a View

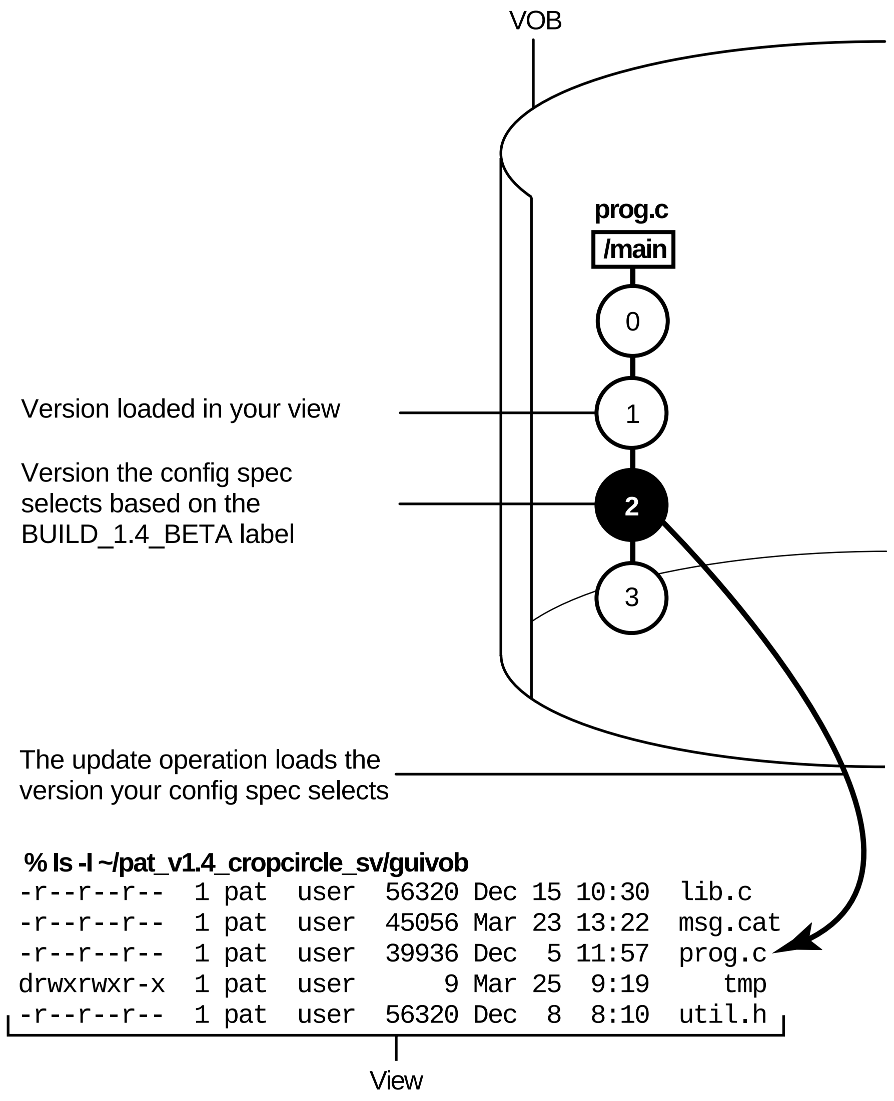
When you start an update operation, ClearCase compares the version of the elements loaded in the view with the version the config spec selects in the VOB. If the config spec selects a version in the VOB that is different from the version loaded in your view, ClearCase copies the version from the VOB into your view (Figure 7). 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 your view's load rules 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).

Figure 7 The Update Operation


VOB
prog.c
/main
0
1
2
3
Version loaded in your view
Version the config spec selects based on the BUILD_1.4_BETA label
The update operation loads the version your config spec selects
View

```
% ls -l ~/pat_v1.4_cropcircle_sv/guivob
-r--r--r--  1 pat  user  56320 Dec 15 10:30  lib.c
-r--r--r--  1 pat  user  45056 Mar 23 13:22  msg.cat
-r--r--r--  1 pat  user  39936 Dec  5 11:57  prog.c
drwxrwxr-x  1 pat  user      9 Mar 25  9:19     tmp
-r--r--r--  1 pat  user  56320 Dec  8  8:10  util.h
```

4.3 Unloading Elements

If a view's config spec 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 on removing load rules, see *Adding or Modifying Load Rules* on page 21.
- ➤ 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 file's current state:

- ➤ 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.

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**.

5 Working On a Team

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 GUI 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, a style of working in which teams use the same set of source files for different, concurrent development projects:

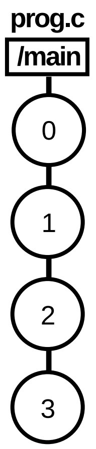
- ➤ Version trees
- ➤ Working on branches
- ➤ Merging
- ➤ Sharing control of a branch in an environment using Rational ClearCase MultiSite

(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.)

5.1 The Version Tree

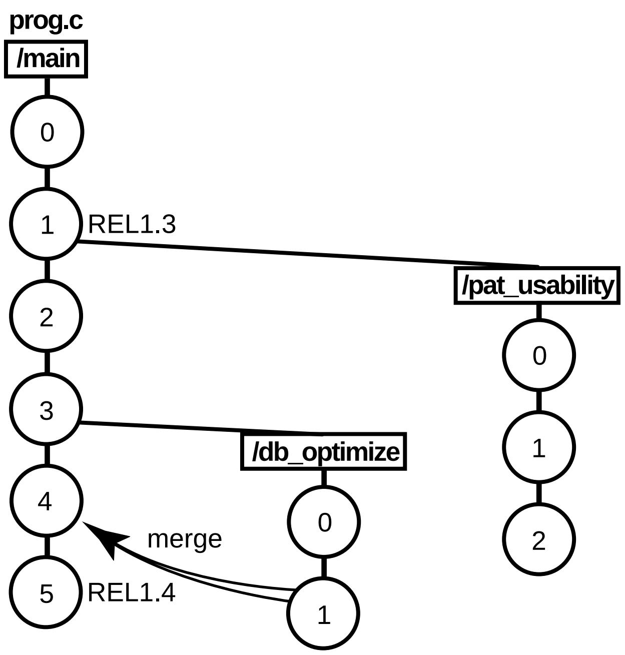
Each time you revise and check in an element, ClearCase creates a new version of the element in the VOB. Throughout this part of the book, this linear progression has been illustrated with a graphic similar to Figure 8.

Figure 8 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 until you merge. In Figure 9, **main**, **pat_usability**, and **db_optimize** are branches being used to develop different releases of the file element **prog.c** concurrently.

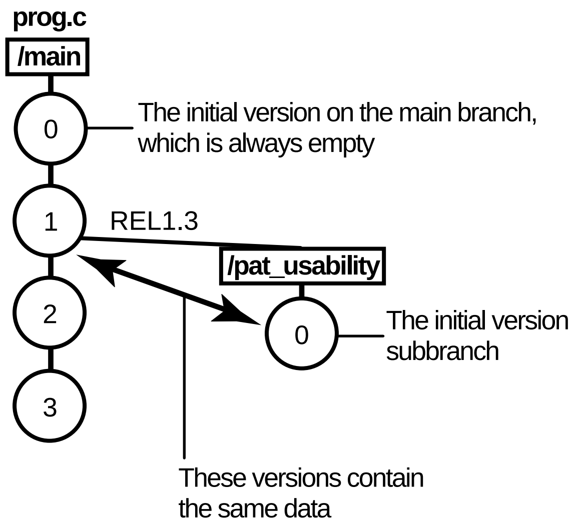
Figure 9 Version Tree of a File Element



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 10). (The initial version on the **main** branch contains no data. For more information, see *Excluding Elements from Loading* on page 22.)

Figure 10 The Initial Version on a Subbranch


prog.c
/main
0
The initial version on the main branch, which is always empty
1
REL1.3
/pat_usability
2
0
The initial version subbranch
3
These versions contain the same data

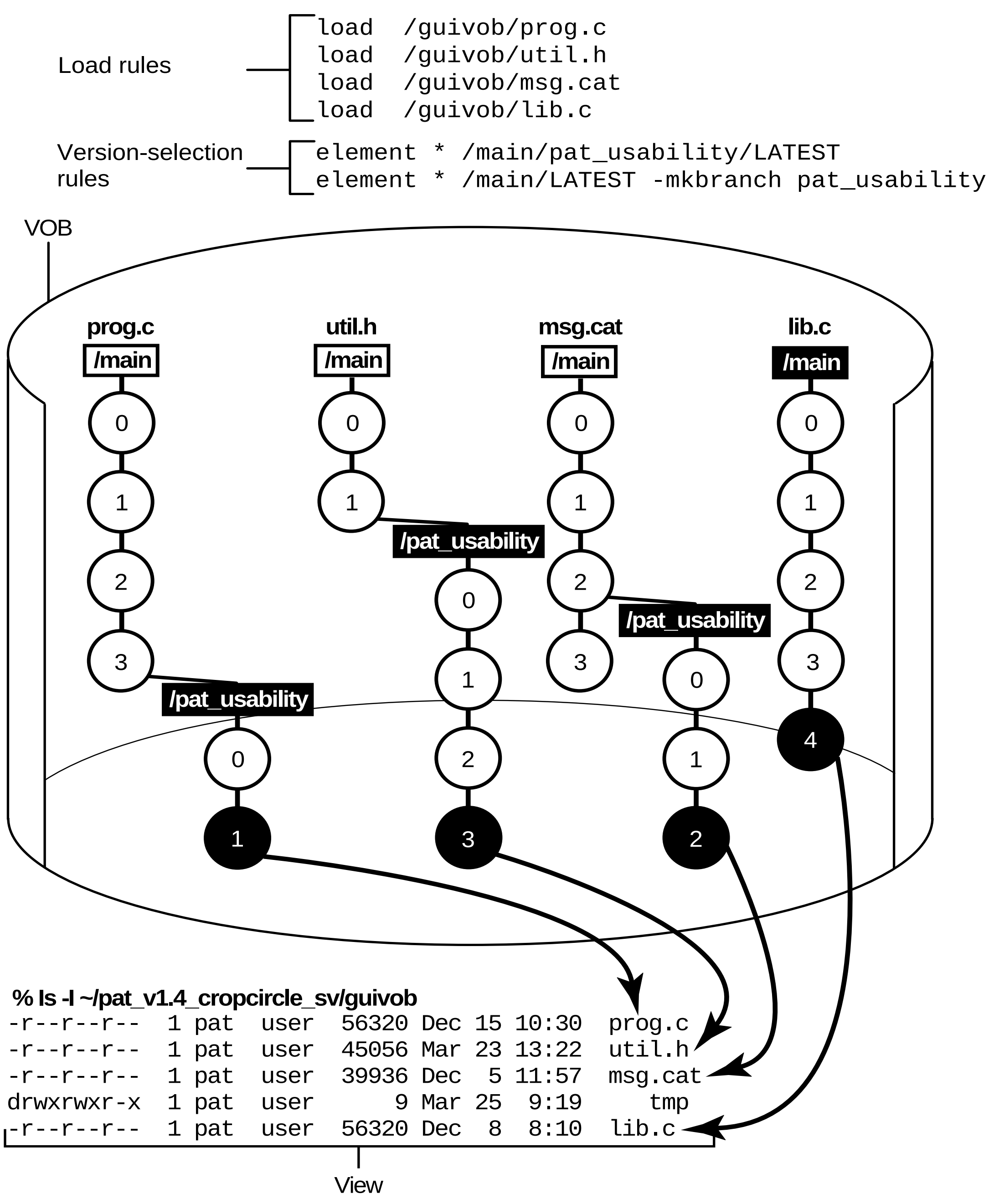
5.2 Working on Branches

Your organization's policies 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 this information:

- The versions to select in the development project's specific branch. As Figure 11 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 11, 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 development project's branch (if it doesn't 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.

Figure 11 Elements Have Common Branches



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

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

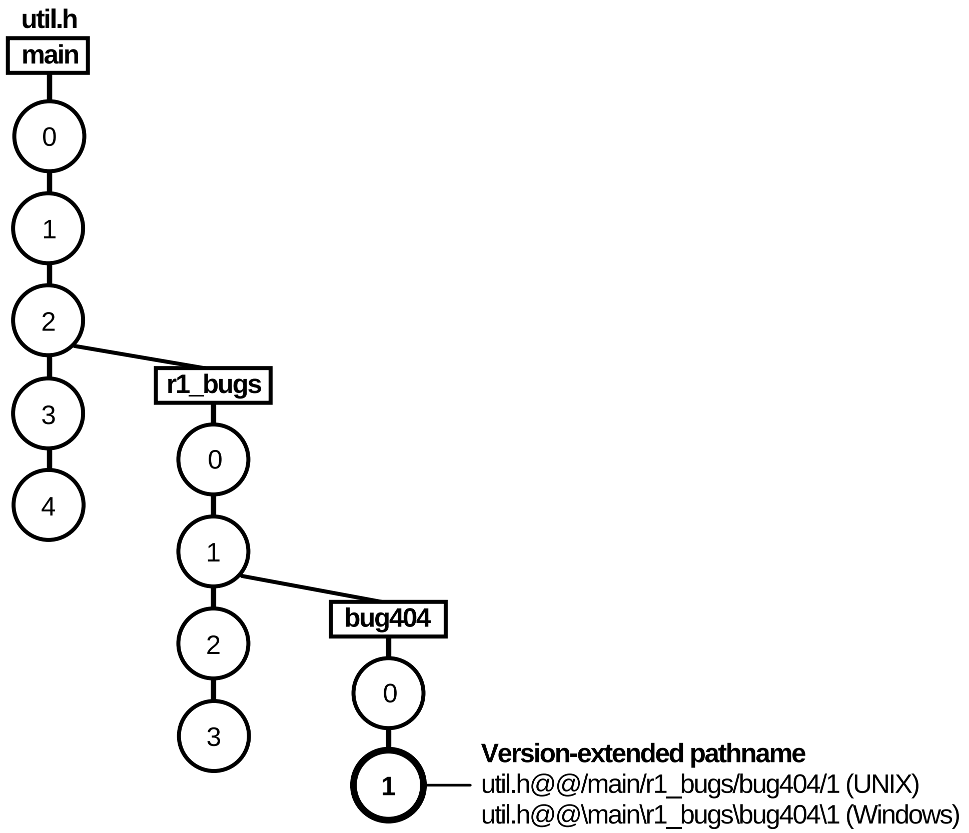
```
% ls -l ~/pat_v1.4_cropcircle_sv/guivob
-r--r--r--  1 pat  user  56320 Dec 15 10:30  prog.c
-r--r--r--  1 pat  user  45056 Mar 23 13:22  util.h
-r--r--r--  1 pat  user  39936 Dec  5 11:57  msg.cat
drwxrwxr-x  1 pat  user      9 Mar 25  9:19     tmp
-r--r--r--  1 pat  user  56320 Dec  8  8:10  lib.c
```

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

The Version-Extended Pathname

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 12).

Figure 12 Version-Extended Pathnames


util.h
main
0
1
2
3
4
r1_bugs
0
1
2
3
bug404
0
1
Version-extended pathname
util.h@@/main/r1_bugs/bug404/1 (UNIX)
util.h@@\main\r1_bugs\bug404\1 (Windows)

From a command-line interface, you can use version-extended pathnames 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 pathnames.

For a full description of the syntax for version-extended pathnames, see the **pathnames_ccase** reference page in the *Command Reference*.

5.3 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 versioning 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:

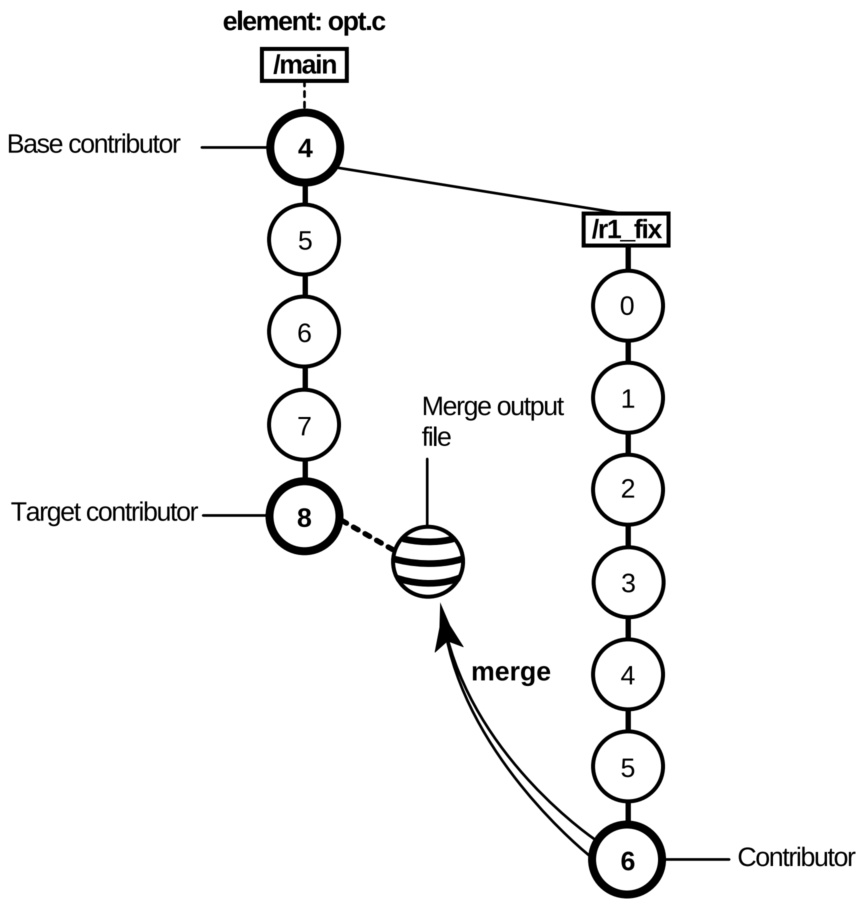
- ➤ Merging all changes made on a single subbranch (see *Scenario: Merging All Changes Made on a Subbranch* on page 65)
- ➤ Merging selectively from a single subbranch (see *Scenario: Selective Merge from a Subbranch* on page 67)
- ➤ Removing the contributions of some versions on a single subbranch (see *Scenario: Removing the Contributions of Some Versions* on page 69)
- ➤ Recording merges that occur outside ClearCase (see *Recording Merges That Occur Outside ClearCase* on page 71)

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 ClearCase merge algorithm uses the following files during a merge (Figure 13):

Figure 13 Versions Involved in a Typical Merge



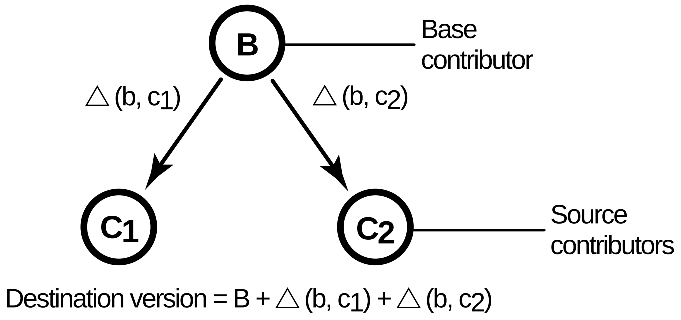
- Contributors, which are typically one version from each branch you are merging. (You can merge up to 15 contributors.) You specify which versions are contributors.
- The base contributor, which is typically the closest common ancestor of the contributors. (For selective merges, subtractive merges, 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 62.
- 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.

- 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 14).
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 14 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 contributor's modified section to the output file:

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

  (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]
  ============
  ```

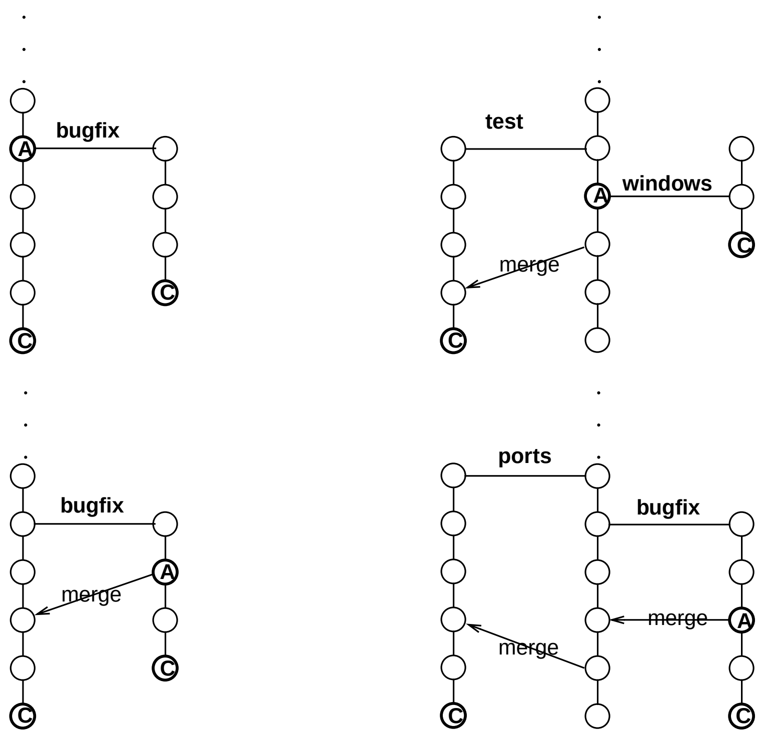
  Be sure to verify that the changes you accept produce consistent merged output. For example, after performing a merge involving file **util.h**, you can compare files **util.h.contrib** (which contains its previous contents) and the new **util.h** (which contains the merge output).

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 element's version tree, 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 15 illustrates common cases of merging. If no merges have been performed in the element, the actual common ancestor (**A**) of the contributors (**C**) in the version tree is selected to be the base contributor.

Figure 15 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 a version's hyperlinks, 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 implements directory merges using VOB hard links. You can use the **ln** and **rmname** commands to perform full or partial merges manually. See the **ln** and **rmname** reference pages 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 16).

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. Set a dynamic view or change directories to a snapshot view. The view must select the target version, which in Figure 16 is **opt.c@@/main/8**.

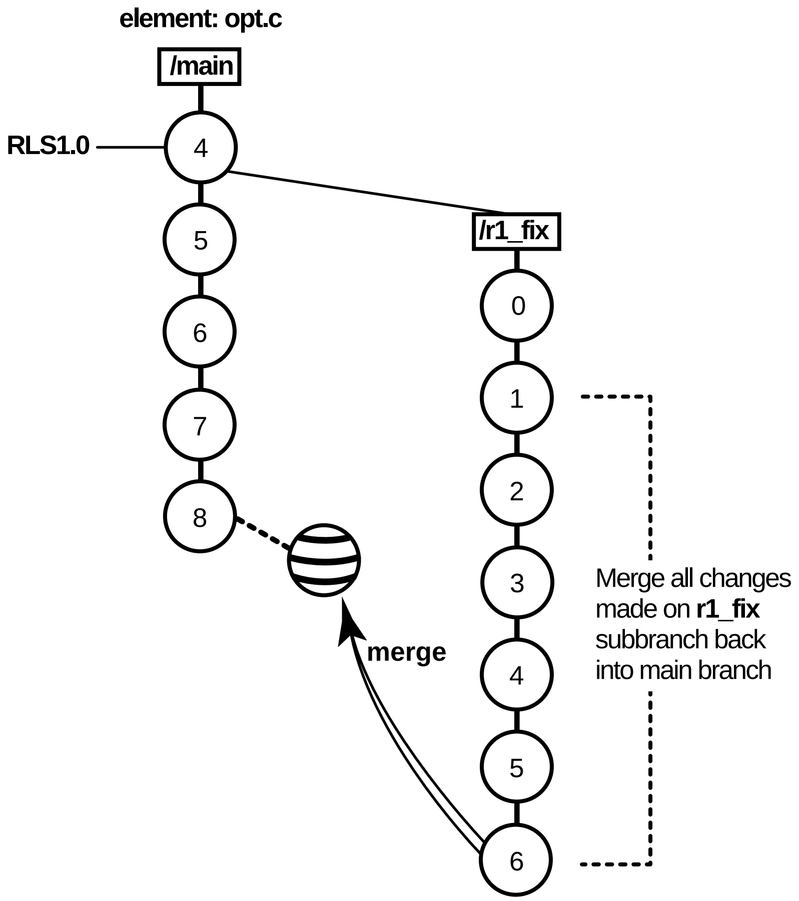
2. If the target version is checked out to your view for other revisions, create a pre-merge checkpoint by checking it in. To make it easier to find this checkpoint, consider labeling the version.

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 16, 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.

Figure 16 Merging All Changes from a Subbranch


element: opt.c
/main
RLS1.0
4
5
6
7
8
/r1_fix
0
1
2
3
4
5
6
merge
Merge all changes made on **r1_fix** subbranch back into main branch

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

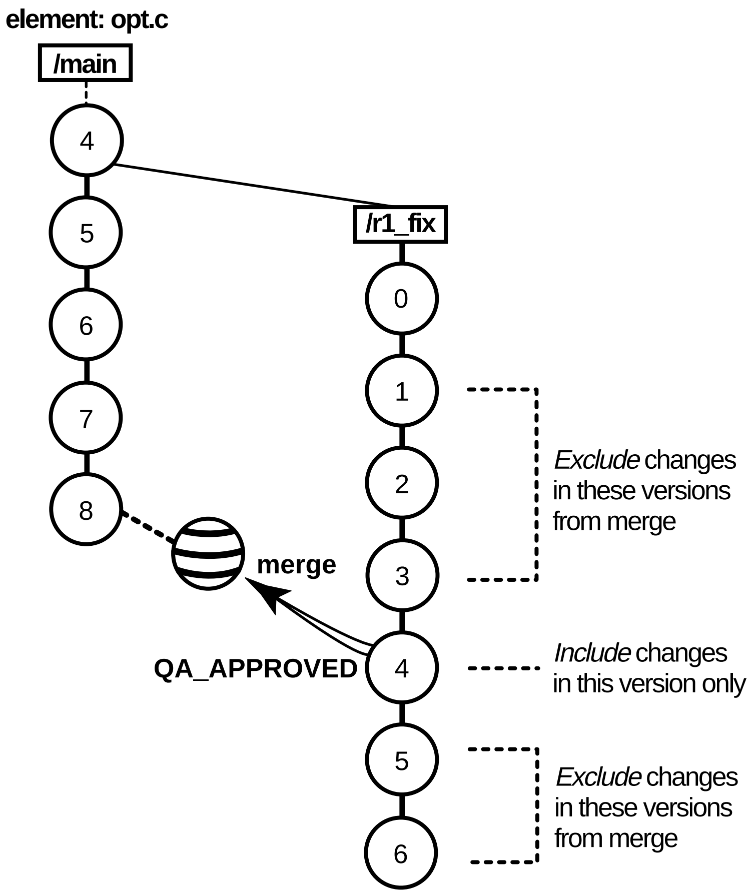
For detailed information about completing this task, see the **findmerge** reference page in the *Command Reference* or ClearCase online help:

From a ClearCase host, type **hyperhelp cc_main.hlp&**

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 17).

Figure 17 Selective Merge from a Subbranch



element: opt.c
/main
4
5
6
7
8
merge
/r1_fix
0
1
2
3
4
5
6
QA_APPROVED
Exclude changes in these versions from merge
Include changes in this version only
Exclude changes in these versions from merge

It's 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. Set a dynamic view or change directories to a snapshot view. 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.

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.

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 on using Diff Merge, refer to the online 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 of a version’s data has been merged, not parts of it.

Getting More Information

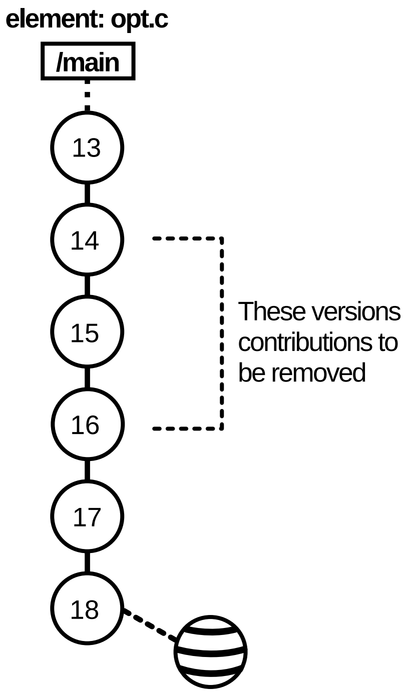
For detailed information about completing this task, see the **merge** and **version_selector** reference pages in the *Command Reference* or ClearCase online help:

From a ClearCase host, type **hyperhelp cc_main.hlp&**

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 18).

Figure 18 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. Set a dynamic view or change directories to a snapshot view. 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 18, 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.

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 on using Diff Merge, refer to online 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 ClearCase online help:

From a ClearCase host, type **hyperhelp cc_main.hlp&**

Recording Merges That Occur Outside ClearCase

You can merge versions of an element manually or with any available analysis and editing tools. To update an element's version tree with a merge that occurs outside ClearCase, check out the target version, perform the merge with your own tools, and check it back in. 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've drawn 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 ClearCase online help:

From a ClearCase host, type **hyperhelp cc_main.hlp&**

5.4 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. You do not need to 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 that replica's site 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 "raleigh".
Current replica is "lexington".
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 check out the branch immediately, using a nonmastered checkout. 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

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: doc_lex@/doc
   ...
   ```

   In this example, your current replica is **lexington** in the VOB family **/doc**. The output of the **describe** command shows that **lexington** 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

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.

6 Other Tasks

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
➤ Moving views
➤ Regenerating a snapshot view's **.view.dat** file
➤ Regenerating **.ccase_svreg**
➤ Accessing views and VOBs across platform types

6.1 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.

Your view's *version-selection rules* determine the element's branch on which the first version 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-pathname*
     cleartool mkelem *file-element-pathname*

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 a File or Directory to Source Control* and the **mkelem** reference page in the *Command Reference*.

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

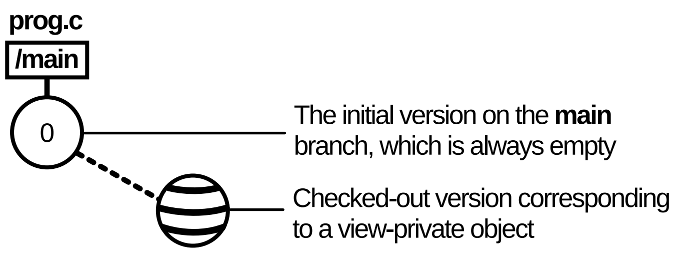
➤ Using **mkelem** with no arguments checks out the element. Any view-private data that corresponds to the element pathname remains in your view only and is added to version 1 in the VOB when you check in (Figure 19).

- Using **mkelem –ci** checks in the element, using any existing view-private data that corresponds to the element pathname as the content for version 1. Your view's config spec 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-pathname* exists, it is moved aside to a **.keep** file.
- (Replicated VOBs only) Using **mkelem –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** 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 you check in the element's parent directories and check in the file or directory.

Figure 19 Creating an Element


prog.c
/main
0
The initial version on the **main** branch, which is always empty
Checked-out version corresponding to a view-private object

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 a VOB's element types.) The element type must already exist in the VOB in which you are creating the new element, or must exist in the Admin VOB hierarchy associated with the VOB in which you are creating the new element. A **mkelem –eltype directory** command is equivalent to a **mkdir** command.

If you do not specify an element type on the command line, **mkelem** determines one by using the magic files to perform file-typing. This involves matching patterns in the new element's name (and examining the existing view-private file with that name, if one exists; see *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. (See the **checkout** reference page in the *Command Reference* for details.)

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 DO. The process is the same as for a shareable DO, 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 DO to a shareable DO, then checks it in.

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

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/LATEST -mkbranch gopher_port
```

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

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

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

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

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

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

For example, **msg.c@@** references an element, whereas **msg.c** 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 for an element's versions (data containers) is allocated in the storage pools that **mkelem** assigns. You can change pool assignments with the **chpool** command.

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're 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*.

6.2 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 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 an element's version tree. 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 *pname target-pname*

3. Check in the parent directory.

6.3 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 element’s ClearCase history.

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 your view's config spec 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 *pathname...*

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-pathname*

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**.

6.4 Moving Views

This section discusses the following tasks:

- ➤ Changing the physical location of a snapshot view’s directory tree
- ➤ 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 snapshot view’s directory tree of loaded elements and view-private files. You can move the view to a different workstation, but the workstation must run a UNIX operating system.

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

To Find the Location of the View Storage Directory

Enter the following command:

cleartool lsview –long *view-tag*

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

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. 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's **view_server** process 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.

6.5 Regenerating a Snapshot View's .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-pathname*

   For example:

   Perl /usr/atria/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.

6.6 Regenerating .ccase_svreg

When you create 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 inadvertently delete or corrupt this file, you must regenerate information in **.ccase_svreg** for each snapshot view that you use. To do so, update the view with either of the following commands:

➤ **cleartool update**
➤ **cleartool update –print**

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

6.7 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 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 view's behavior.

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). Even though 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: it does not create symbolic links if the load rules encounter a VOB symbolic link, and 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 93.

2. Import the UNIX view's view-tag 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 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 online help.

Working in a Snapshot View While Disconnected from 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 ClearCase Web interface. Ask your ClearCase administrator whether the ClearCase Web interface 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 online help.

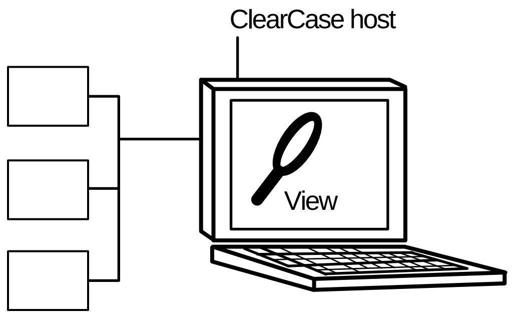
A.1 Setting Up a View for Your Hardware Configuration

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 20).

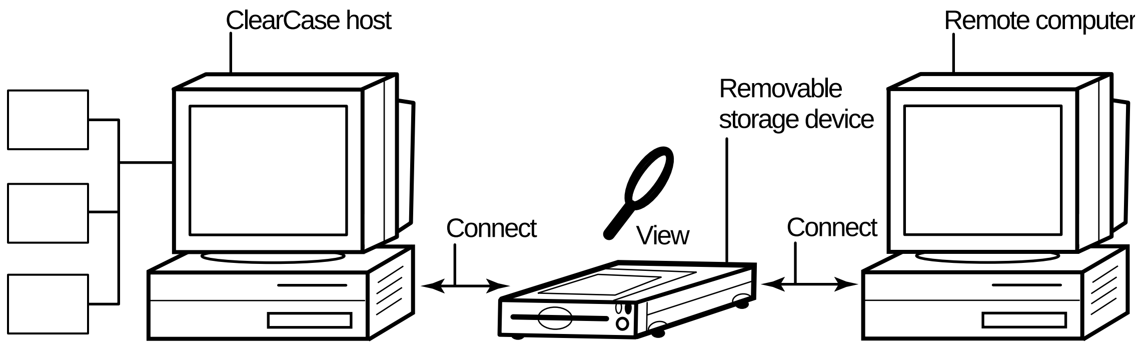
Figure 20 View on a Laptop



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 21).

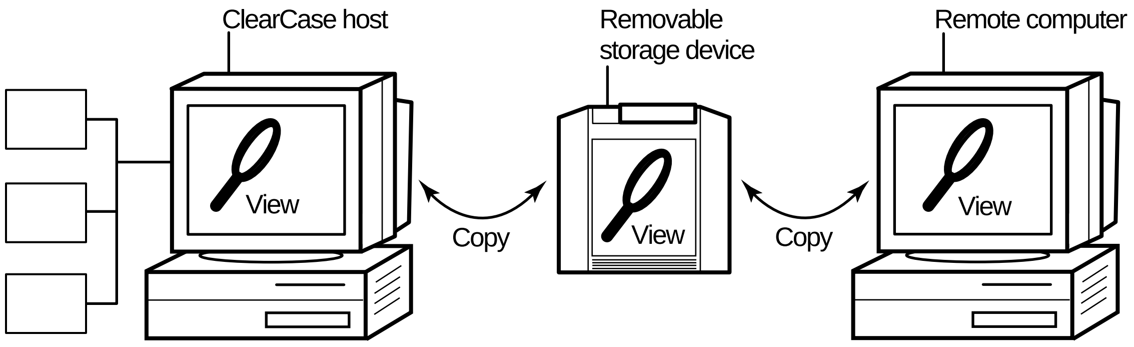
Figure 21 View On a Removable Storage Device



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 22).

Figure 22 Copy the View



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

Under the Hood: Location of the View Storage Directory 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 a view's **view_server** process runs on the host that 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.

A.2 Preparing the View

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

- Update the view to establish a checkpoint. (For information on updating the view, see *Updating the View* on page 103.)

- ➤ Check out the files you expect to modify. After you’re disconnected from the network, you cannot check out files, although there are workarounds. (See *Hijacking a File* on page 98.)

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.

A.3 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

A.4 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: How ClearCase Determines Whether a File is Hijacked* on page 103.

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're ready, check in your changes.
- ➤ Undo the hijack. For more information, see *To Undo a Hijack* on page 103.

To Hijack a File

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

chmod +w prog.c

Finding 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-pathname* **–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.

A.5 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.

A.6 Using the Update Tool

When you're 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

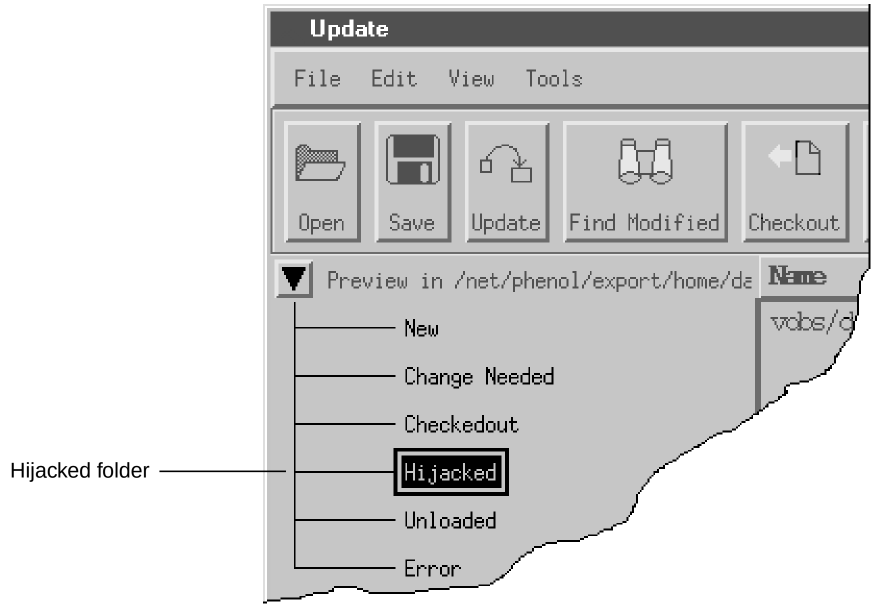
1. Enter the following command:

   cleartool update –graphical *snapshot-view-pathname*

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

3. If any hijacked files are in your view, the ClearCase Snapshot View Update window displays a folder in the left pane titled **Hijacked** (Figure 23). Select **No** for the option asking whether you want to check out the hijacked files now.

Figure 23 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 ClearCase Snapshot View Update window, click a hijacked file.
2. Click **Tools** > **Compare with old**. For information on using the Diff Merge tool, see the online help.

To Check Out a Hijacked File

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

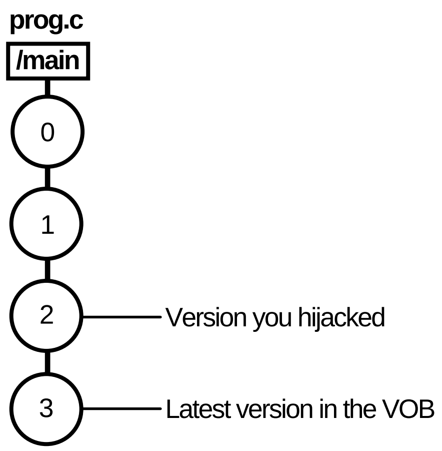
1. In the right pane of the ClearCase Snapshot View Update window, click a hijacked file.
2. Click **Tools** > **Checkout**.
3. ClearCase treats a checked-out hijacked file as it does any other checkout.

When you're ready, you can check in the file.

Merging the Latest Version to a Hijacked File

If you're 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 24), ClearCase prevents you from checking in the file.

Figure 24 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 pathname*. For more information, see the **pathnames_ccase** reference page in the *Command Reference*.

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 ClearCase online help. 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 ClearCase 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 103.

Under the Hood: How ClearCase Determines Whether a File is Hijacked

To keep track of file modifications in a snapshot view, ClearCase stores a loaded file's size and last-modified time stamp (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 a non-checked-out file's read-only permission alone 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

See *Updating the View* for more information.

Updating the View

1. Enter the following command:

```
cleartool update –graphical snapshot-view-pathname
```

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