INTRODUCTION TO CLEARCASE

Release 4.0 and later

Windows/UNIX Edition

Rational
unifying **software** teams

800-012607-000

Introduction to ClearCase
Document Number 800-012607-000 December 1999
Rational Software Corporation 20 Maguire Road Lexington, Massachusetts 02421

IMPORTANT NOTICE

Technical Acknowledgments

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

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

Rational
unifying **software** teams

Contents

Figures

Preface

Rational ClearCase and Rational ClearCase MultiSite provide a comprehensive solution for software configuration management and distributed development.

Rational Unified Change Management (UCM) provides a best practices approach to comprehensive change management, by tightly integrating ClearCase and ClearCase MultiSite with Rational ClearQuest, a change request management product, and by providing an out-of-the-box software development and change management process for using these products.

About This Manual

This manual provides basic descriptions of ClearCase, ClearCase MultiSite, ClearQuest, and Unified Change Management. It also provides an overview of how to deploy these products in an organization, from planning, site preparation, and installation through setting up, working on, and managing software development projects. Where appropriate, the manual refers you to specific locations in ClearCase, ClearCase MultiSite, and ClearQuest documentation for detailed information about individual procedures and concepts.

User Roles, the ClearCase Documentation Set, and This Manual

The documentation for ClearCase consists of printed and online task-oriented information, supporting ClearCase users acting in the following roles:

- **project manager** — defines, implements, and manages the objects, policies, and processes of a software development project
- **developer** — makes changes to the software configuration (that is, the files and directories) that belong to a software development project

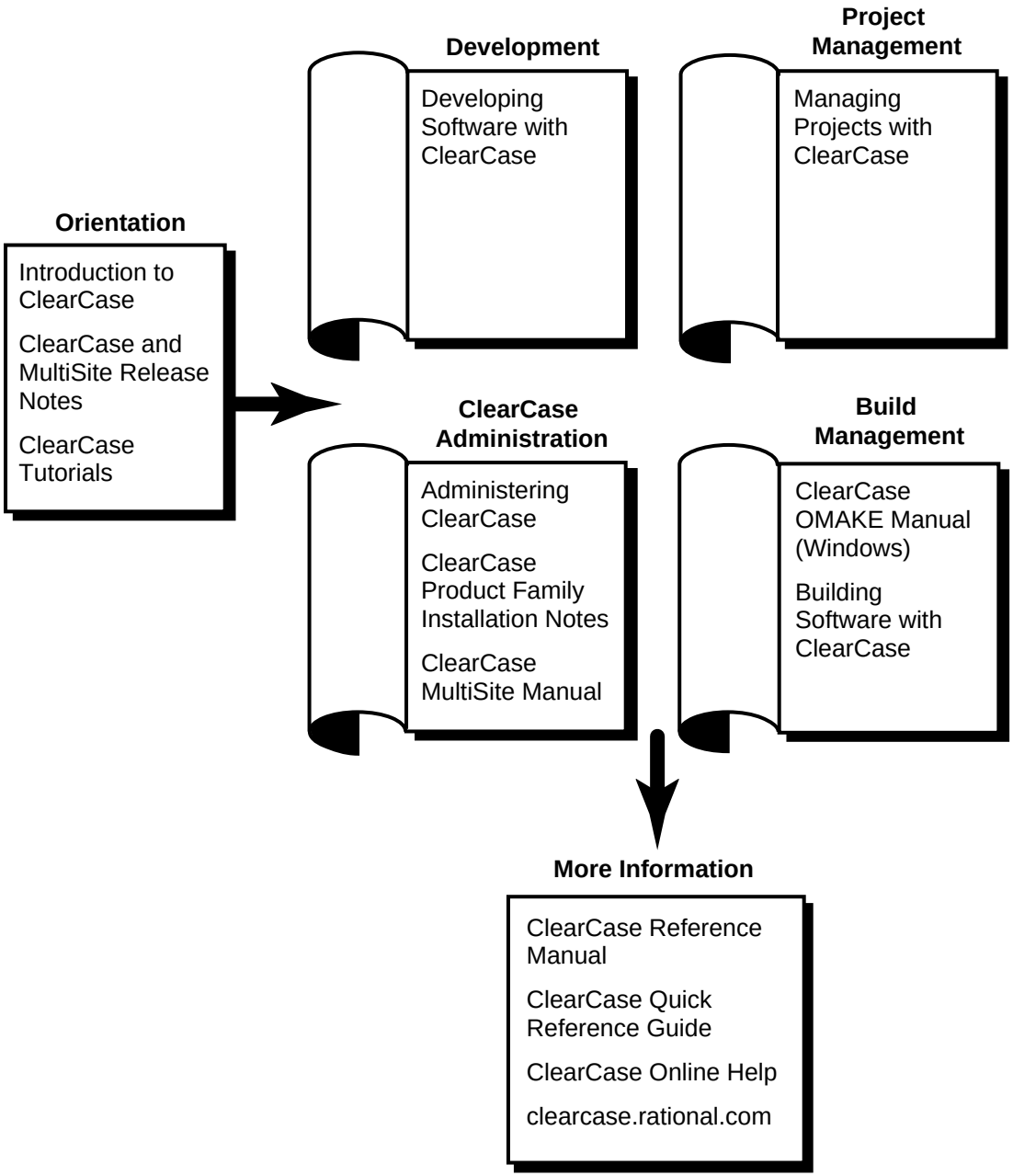
- **integrator** (also called build engineer or release engineer) — builds and integrates the products of a software development project
- **administrator** — configures and maintains the ClearCase infrastructure, including ClearCase VOBs, views, servers, and clients, for part or all of your organization

The information in this manual applies to these roles as follows:

- Chapter 1, *ClearCase, ClearQuest, and Unified Change Management*, contains information of interest to all roles.
- Chapter 2, *Planning for and Installing ClearCase*, contains information of interest to administrators and project managers; also, the instructions for installing ClearCase on your computer applies to all users.
- Chapter 3, *Setting Up a Software Project in ClearCase*, contains information of interest to project managers.
- Chapter 4, *Developing and Building Software with ClearCase*, contains information of interest primarily to developers and integrators; also, the information about the development process in ClearCase could be of interest to project managers.
- Chapter 5, *Managing Software Projects with ClearCase*, contains information of interest primarily to project managers.
- The glossary that appears at the end of this manual contains information of interest to all roles.

The *ClearCase Documentation Roadmap* that appears in the next section shows how the ClearCase documentation set is organized to support these roles.

ClearCase Documentation Roadmap



Typographical Conventions

This manual uses the following typographical conventions:

- *ccase-home-dir* represents the directory into which the ClearCase Product Family has been installed. By default, this is **/usr/atria** on UNIX and **C:\Program Files\Rational\ClearCas**e on Windows.
- *attache-home-dir* represents the directory into which ClearCase Attache has been installed. By default, this directory is **C:\Program Files\Rational\Attache**, except on Windows 3.*x*, where it is **C:\RATIONAL\ATTACHE**.
- **Bold** is used for names the user can enter; for example, all command names, file names, and branch names.
- *Italic* is used for variables, document titles, glossary terms, and emphasis.
- `A monospaced font` is used for examples. Where user input needs to be distinguished from program output, **bold** is used for user input.
- Nonprinting characters are in small caps and appear as follows: **<EOF>**, **<NL>**.
- Key names and key combinations are capitalized and appear as follows: F1, SHIFT, CTRL+G.
- [ ] Brackets enclose optional items in format and syntax descriptions.
- { } Braces enclose a list from which you must choose an item in format and syntax descriptions.
- | A vertical bar separates items in a list of choices.
- ... In a syntax description, an ellipsis indicates you can repeat the preceding item or line one or more times. Otherwise, it can indicate omitted information.

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

  lsc·heckout

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

Online Documentation

The ClearCase graphical interface includes an online help system.

There are three basic ways to access the online help system: the **Help** menu, the **Help** button, or the F1 key. **Help→Contents** provides access to the complete set of ClearCase online documentation. For help on a particular context, press F1. Use the **Help** button on various dialog boxes to get information specific to that dialog box.

ClearCase also provides access to full "reference pages" (detailed descriptions of ClearCase commands, utilities, and data structures) with the **cleartool man** subcommand. Without any argument, **cleartool man** displays the **cleartool** overview reference page. Specifying a command name as an argument gives information about using the specified command. For example:

cleartool man *(display the cleartool overview page)*

cleartool man man *(display the cleartool man reference page)*

cleartool man checkout *(display the cleartool checkout reference page)*

ClearCase's **–help** command option or **help** command displays individual subcommand syntax. Without any argument, **cleartool help** displays the syntax for all **cleartool** commands. **help checkout** and **checkout –help** are equivalent.

cleartool lsprivate –help

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

Additionally, the online *ClearCase Tutorial* provides a step-by-step tour through ClearCase's most important features. To start the tutorial:

- On Windows, choose **Tutorial** from the **Getting Started** tab of ClearCase Home Base.
- On UNIX, type **hyperhelp cc_tut.hlp**.

Technical Support

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

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

1 ClearCase, ClearQuest, and Unified Change Management

This chapter contains short summaries of ClearCase, ClearCase MultiSite, and ClearQuest, and a description of how Rational Unified Change Management (UCM) integrates these products to provide an out-of-the-box software development and change management process.

1.1 ClearCase

Rational ClearCase® is a *configuration-management* system designed to help software development teams track the files and directories used to create software. ClearCase enables you to manage the development and build process and to enforce your site-specific development policies.

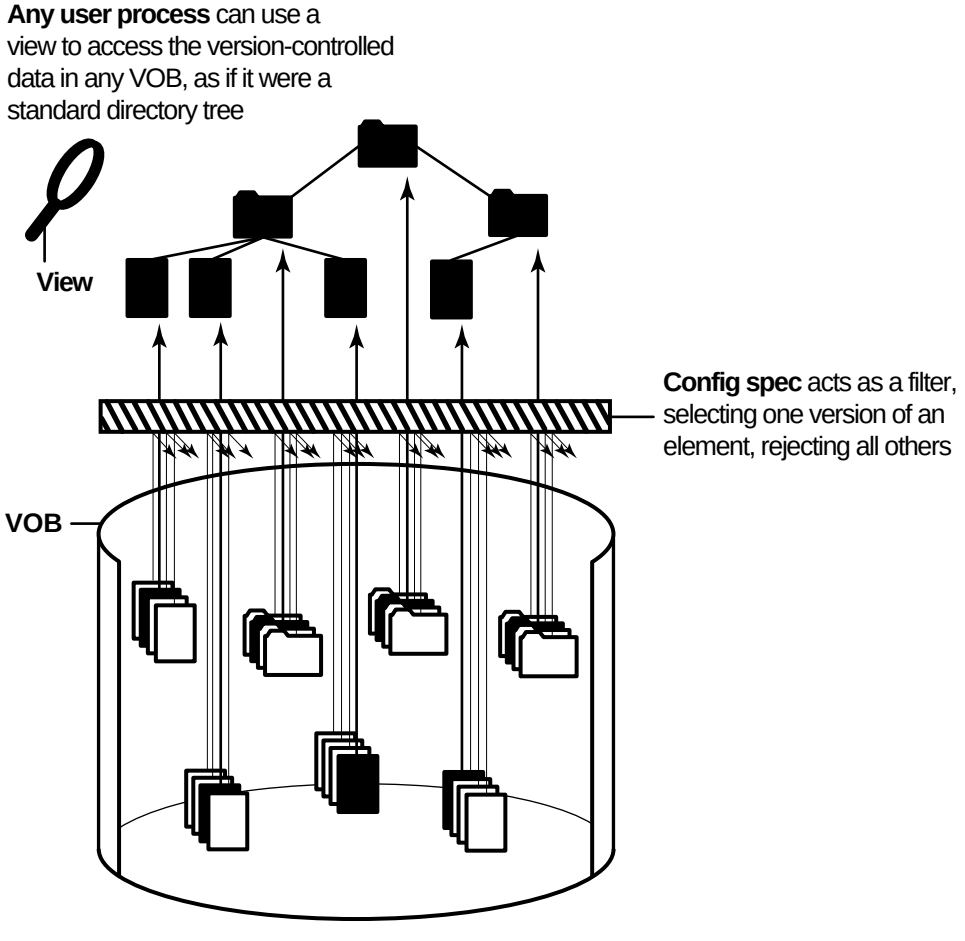
ClearCase is specifically designed to support parallel development, whether you are simply isolating the work of one developer from others on a small team, developing multiple releases in parallel using different teams, or sharing a source code base between multiple teams at geographically distributed sites.

ClearCase enables you to re-create the source base from which a software system was built, allowing it to be rebuilt, debugged, and updated—all without interfering with other development work.

In ClearCase, files and directories, or *elements*, are stored in a repository called a *versioned object base* or *VOB*. A *version* is a particular revision of a file or directory element.

You access and change elements using a *view*. A VOB contains all versions of a particular set of elements; a view selects a specific version of each element using a set of rules called a configuration specification (or *config spec*). The result is that when accessed through a view, a VOB looks just like an ordinary file-system directory tree. (See Figure 1.)

Figure 1 Accessing a VOB Using a View



Like many configuration management systems, ClearCase uses a checkout-edit-checkin model to manage software changes. When you check out a file, ClearCase creates an editable copy, or

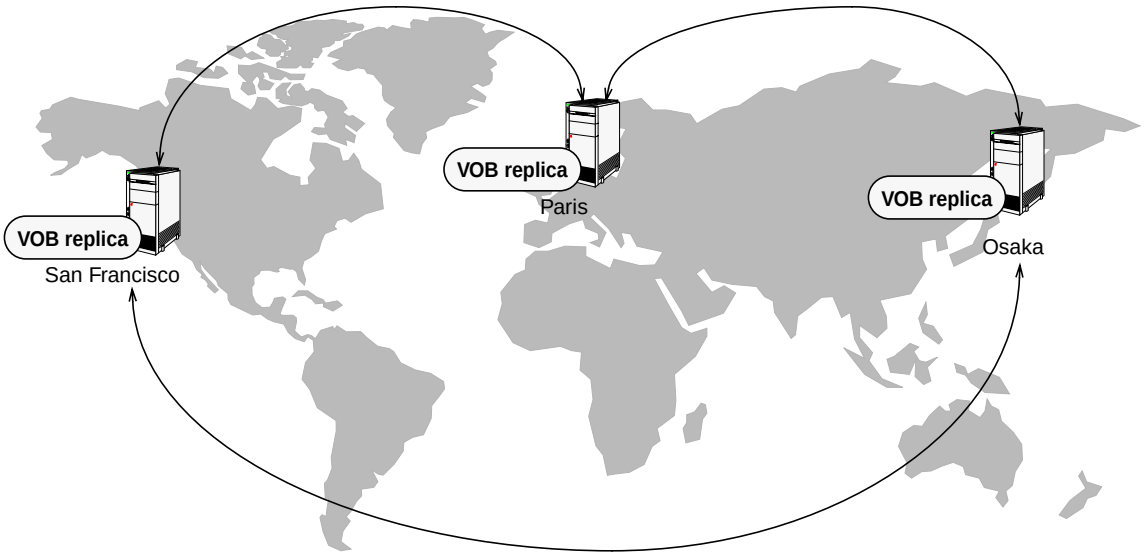
checked-out version, in your view. When you check in a file, ClearCase creates a new, permanent version of the file in the VOB.

1.2 ClearCase MultiSite

Rational ClearCase MultiSite® extends ClearCase by supporting parallel software development and software reuse across geographically distributed project teams.

ClearCase MultiSite enables developers at different locations to use the same VOB. Each site has its own copy, or *replica*, of that VOB. The set of replicas for a particular VOB is called a *VOB family*. At any time, a site can propagate the changes made in its own VOB replica to the other members of the VOB family, using either an automatic or manual synchronization process.

Figure 2 ClearCase MultiSite VOB Family



This manual discusses ClearCase MultiSite only where it applies to a given ClearCase operation or concept. See *ClearCase MultiSite Manual* for details about configuring, using, and administering ClearCase MultiSite.

1.3 ClearQuest

Rational ClearQuest® is a change request management application that allows you to track change requests for your products. Using ClearQuest, you can submit change requests, view and modify existing change requests, and create and run user- or site-specific queries and reports to determine the current state of your project.

In ClearQuest, change requests are stored as *records* in a ClearQuest database. Each record consists of all the data related to that record. ClearQuest supports different types of records for different projects and uses. For example, you might have record types for enhancements, defects, and activities, each with unique fields and data requirements.

A *schema* refers to all attributes that define a ClearQuest database. ClearQuest provides default schemas and allows you to create customized schemas.

ClearQuest records move through a pattern, or lifecycle, from submission through resolution. In ClearQuest, each stage in this lifecycle is called a *state*, and each movement from one state to another is called a *state transition*.

This manual discusses ClearQuest only where it applies to a given ClearCase operation or concept. See the ClearQuest documentation set for details about configuring, using, and administering ClearQuest.

1.4 Unified Change Management

Rational Unified Change Management (UCM) combines ClearCase and ClearQuest to provide a complete, out-of-the-box, activity-based change management process.

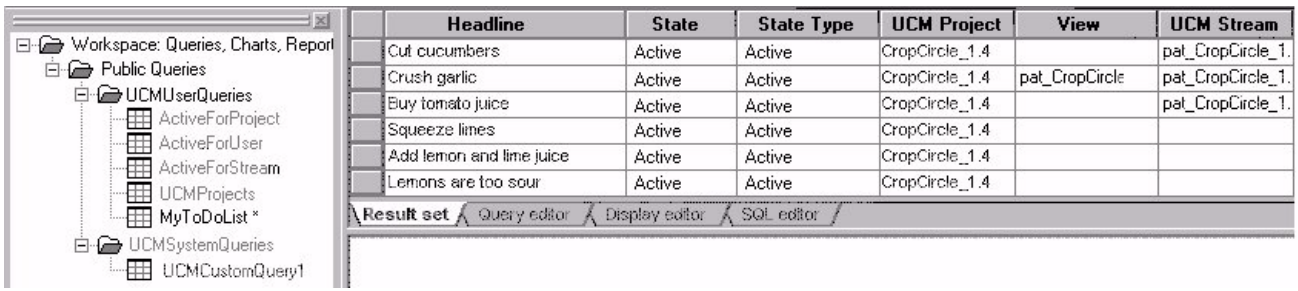
UCM combines ClearCase configuration management capabilities (such as version control, parallel development, build management, and component-based management of directories and files) with ClearQuest change request and activity management capabilities, such as (task management, state transition support, parent/child associations, policy enforcement rules, and extensive querying and reporting).

In UCM, development work is organized into *projects*, which are ClearCase objects that contain the configuration information needed to manage a significant development effort, such as a product release. You use a project to set the policies that govern how developers access and update the set of files and directories used in the development effort.

An *activity* is a ClearCase object that tracks the work required to complete a particular development task.

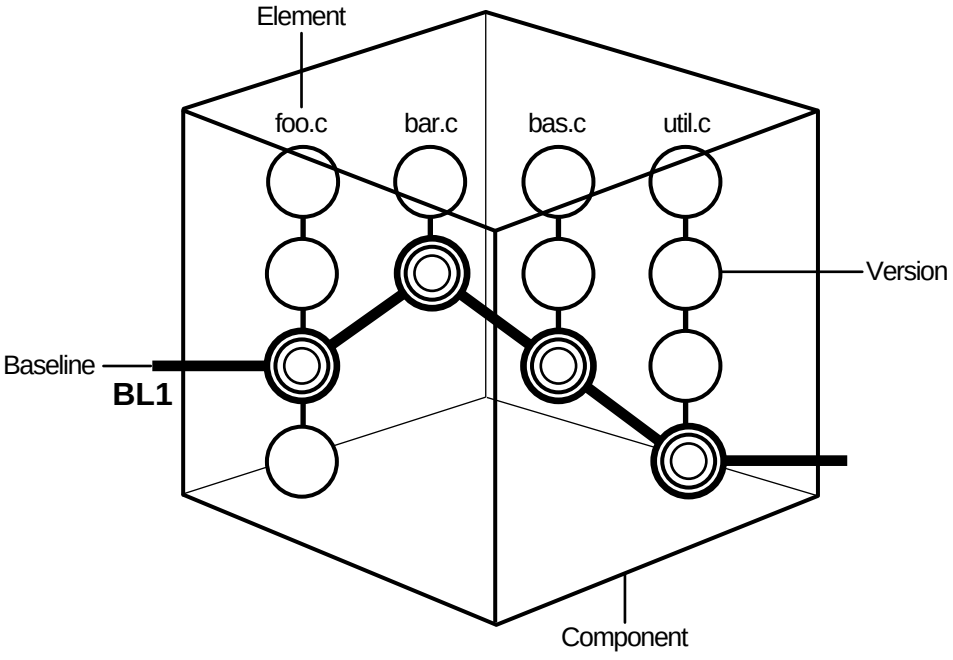
You can associate ClearCase project and activity objects with ClearQuest records. This enables you to attach ClearQuest information such as states and state transitions, user assignments, and parent/child associations to ClearCase projects and activities. Developers can use ClearQuest queries to determine which activities are assigned to them, as shown in Figure 3. Project managers can use ClearQuest queries, reports, and charts to monitor the progress of software development projects.

Figure 3 Using a ClearQuest To-Do List to Find UCM Activities



In UCM, a *component* is a group of related ClearCase directory and file elements, which you develop, integrate, and release together. A *baseline* is a version of a component (see Figure 4). A baseline typically represents a stable configuration for that component, such as a project milestone or release.

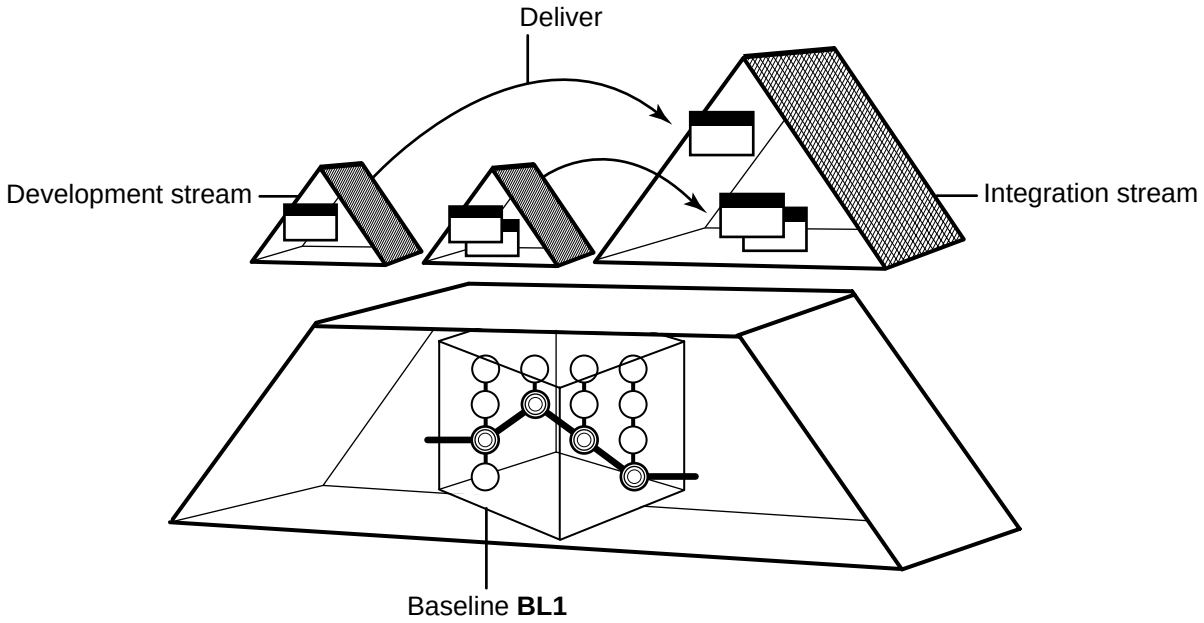
Figure 4 Elements, Components, and Baselines



A UCM project includes a single *integration stream*, which configures ClearCase views that select the latest versions of a project's shared ClearCase elements. When a developer joins a project, ClearCase creates a *development stream* for that developer in the project. Development streams configure the ClearCase views that allow developers to work on the project components in isolation from the rest of the team.

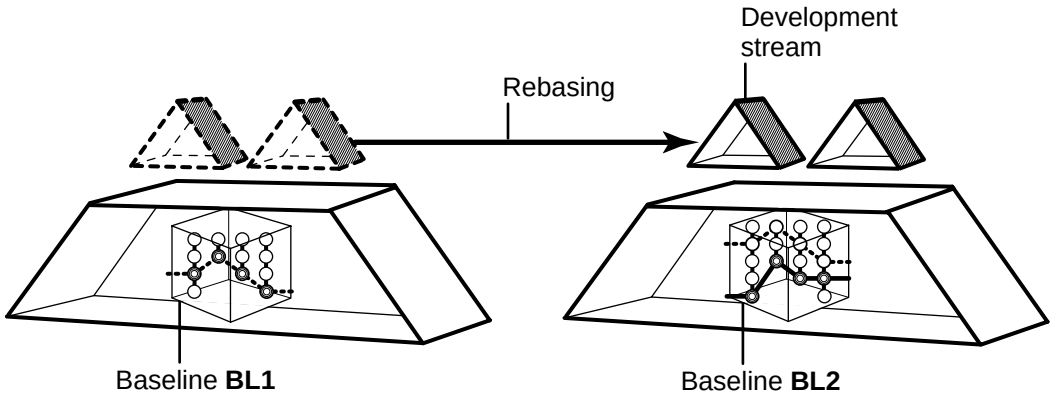
When developers work on a particular activity, all changes to the files and directories in the components are associated with that activity in a *change set*. When developers complete activities, and build and test their work in their private work areas, they share their work with the project team by *delivering* the activities from their development streams to the project's integration stream, as shown in Figure 5.

Figure 5 Delivering Activities from Development Streams to Integration Streams



Periodically, the project manager creates new baselines for the components used by the project. When the project manager recommends a particular baseline, the developers may choose to *rebase* their development streams to use the new baseline, as shown in Figure 6. The new baselines incorporate work that developers have delivered since the last baselines were created.

Figure 6 Rebasing Development Streams



Note that using UCM or ClearQuest is optional. You can choose not to use ClearQuest for change request and activity management and still take advantage of UCM process features. You can choose not to use UCM process features and still associate ClearQuest records with ClearCase versions. Or you can use ClearCase without UCM or ClearQuest.

This section provides only an overview of how UCM integrates ClearCase, ClearCase MultiSite, and ClearQuest. See *Developing Software with ClearCase*, *Managing Software Projects with ClearCase*, and the ClearQuest documentation set for details about the concepts and tasks summarized here.

2 Planning for and Installing ClearCase

This chapter summarizes the basic steps required to install ClearCase, providing a high-level understanding of the installation process. It is not intended to be a complete installation guide. See *ClearCase Product Family Installation Notes* for details about installing ClearCase at your site.

If you are a ClearCase administrator, charged with installing ClearCase at your site, read this chapter.

If you are a project manager, *Planning Issues* contains information about using UCM, ClearQuest, and MultiSite in your organization.

If you are concerned only with installing ClearCase on your client computer, start with *Installing ClearCase on Individual Computers* on page 12.

2.1 Planning Issues

This section describes some of the significant decisions administrators or project managers must make before you can begin installing ClearCase at your site.

Using Unified Change Management or Base ClearCase

UCM is an optional prescribed method of using ClearCase for version control and configuration management. Because UCM is layered on base ClearCase, it is possible to work efficiently in UCM without having to master the details of base ClearCase.

UCM provides the convenience of an out-of-the-box solution; base ClearCase offers the flexibility to implement virtually any configuration management solution that you deem appropriate for your environment.

By default, UCM functionality is included when you install ClearCase. However, this does not prevent you from using base ClearCase functionality, because you can specify that your configuration (specifically, your VOBs and views) not use UCM features.

Chapters 3, 4, and 5 of this manual provide an overview of the differences in configuring, working in, and managing software development projects between base ClearCase and UCM. However, for detailed information:

- See *Managing Software Projects with ClearCase* for information about creating and managing projects using UCM or base ClearCase, including the details about what you must consider for each before installing ClearCase.
- See *Developing Software with ClearCase* for information about how choosing UCM or base ClearCase affects how developers do their work in ClearCase.

Using ClearQuest

In UCM, you can use ClearQuest to provide activity management for your UCM development work, such as assigning and transitioning states for activities, and querying and reporting on activities based on state, user assignment, project, and so on. See *Managing Software Projects with ClearCase* for details about configuring ClearQuest and ClearCase to support UCM activity management.

If you are using base ClearCase functionality instead of UCM, you can associate ClearQuest change requests with ClearCase versions. See the online documentation for the ClearQuest-ClearCase Integration Configuration program for details about configuring this integration.

Using ClearCase MultiSite

Before installing ClearCase MultiSite, you must resolve planning issues, such as which development artifacts to share across sites, how the various sites will access and change those artifacts, how to synchronize sites, and so on.

See *ClearCase MultiSite Manual* for details about planning for and using ClearCase MultiSite.

2.2 ClearCase Site Preparation

The ClearCase administrator (possibly with advice from the ClearCase project manager) decides how to configure the ClearCase installation for the site. The administrator creates a shared release area from which users can install ClearCase on their individual computers. When creating the shared release area, the administrator specifies default values for installation parameters for the individual hosts based on the configuration decisions made for the site.

See READ ME FIRST

The *READ ME FIRST* chapter in *ClearCase and MultiSite Release Notes* contains information you need to know before installing ClearCase at your site:

- Supported platforms and file systems (including Windows/UNIX file access)
- Hardware and software requirements
- Platform-specific information pertaining to installation, such as disk space required, OS patches required, layered software packages required, and so on
- ClearCase and MultiSite patches incorporated into this release
- Issues with upgrading from a previous ClearCase release (both in general and pertaining to this particular release)
- Known issues pertaining to installation

Running ClearCase Site Preparation

The ClearCase Site Preparation program prepares the shared release area from which users can install ClearCase on their computers.

ClearCase Site Preparation configures the site-wide defaults that users see when installing ClearCase on their computers. This simplifies site-wide installation because the ClearCase administrator defines the ClearCase installation parameters for the site only once rather than relying on everyone making the appropriate choices when they install. It also simplifies installation, because users can accept the default values when prompted.

On Windows, running ClearCase Site Preparation requires at least local administrator privileges and often also requires network administrator privileges. On UNIX, running ClearCase Site Preparation requires root privileges.

See *ClearCase Product Family Installation Notes* for detailed information about running ClearCase Site Preparation.

2.3 Installing ClearCase on Individual Computers

To install ClearCase on your computer, go to the release area created by your ClearCase administrator and run the ClearCase Installation program. (On Windows, this is **setup.exe**; on UNIX, it is **install_release**.)

Typically, you should accept the default installation parameters. See your ClearCase administrator before you override any default values.

See *ClearCase Product Family Installation Notes* for detailed information about installing ClearCase.

3 Setting Up a Software Project in ClearCase

This chapter provides an overview of how a project manager sets up a software development project in ClearCase, using either UCM or base ClearCase.

Depending on the complexity of your organization and your software configuration, you may need to create and organize many projects. This chapter describes the basic steps required to create a single project. See Chapter 2 for an overview of the planning issues you need to consider before creating a project in ClearCase.

See *Managing Software Projects with ClearCase* for detailed information about creating and configuring projects in ClearCase, including how to plan for, create, and manage multiple projects.

3.1 Creating a Project in UCM

This section provides an overview how to set up a project in UCM.

Creating a Project VOB

In UCM, each project must be associated with a *project VOB*, or *PVOB*. A PVOB is a special kind of VOB that stores UCM objects, such as projects, activities, and change sets. The PVOB must exist before you can create the UCM objects associated with it.

Organizing Directories and Files into VOBs and Components

Before starting a project, the project manager or ClearCase administrator must map the existing system architecture into UCM *components*. To do this, you create ClearCase VOBs that correspond to the files and directories in the system architecture and then define UCM components whose contents are those VOBs.

For existing ClearCase VOBs, you apply a label to the directories and files in those VOBs to define a starting point for the UCM configuration. You then define UCM components for the existing VOB content based on that label. See *Managing Software Projects with ClearCase* for details about converting existing ClearCase VOBs into UCM components.

For existing architectures currently maintained outside ClearCase, you must import those files and directories into VOBs and then define UCM components for those VOBs as described above. See *Administering ClearCase* for details about importing existing files and directories into ClearCase VOBs.

Creating a Project

In UCM, a *project* is an object in a PVOB that contains the configuration information needed to manage a development effort. A project defines how developers access and update the components used in a particular development effort, including:

- Which *components* and *baselines* are to be used for this project
- How developers are to work both independently of and in conjunction with each other
- How to group development changes into manageable pieces
- Whether to use the ClearQuest integration, and if so, which ClearQuest database to associate with this project

Creating and Assigning Activities

In UCM, an *activity* is an object that tracks the work required to complete a particular development task. As project manager, you decide whether to create activities and assign them to developers as part of setting up your project, or to allow the developers to create their own activities as they do their work.

Using the ClearQuest Integration

If your project uses the ClearQuest integration, UCM activities can be associated with ClearQuest records, enabling you to attach project management information such as states and state transitions, user assignments, policy enforcement rules, parent/child associations to ClearCase activities.

Developers can use a to-do list in ClearQuest to access activities that the project manager or other team members assign to them.

3.2 Setting Up a Project in Base ClearCase

This section provides an overview of how to set up a project in base ClearCase.

Importing Directories and Files into VOBs

Before starting a project, the project manager or ClearCase administrator must import the files and directories that constitute the existing system architecture into ClearCase VOBs. See *Administering ClearCase* for details about importing existing files and directories into ClearCase VOBs.

Applying a Label to the Initial Configuration

A *label* is a user-defined name that can be attached to a version.

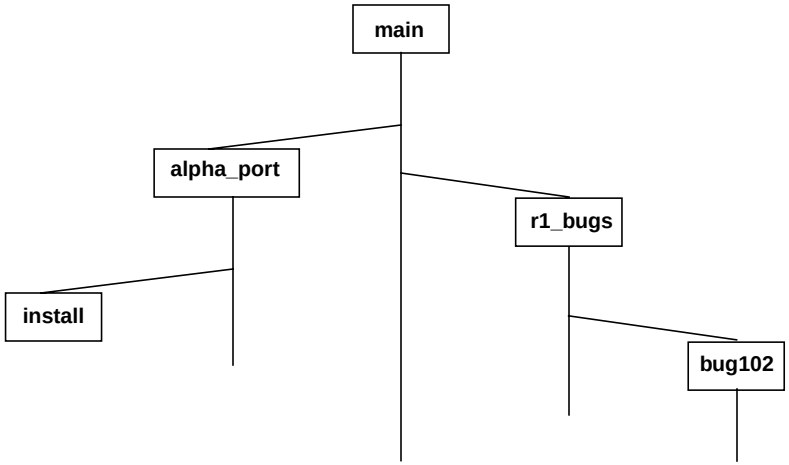
After importing the system configuration into ClearCase VOBs, you can apply a label to the directories and files in those VOBs to define a starting point for your project configuration.

Deciding a Branching and Merging Strategy

Base ClearCase uses branches directly to implement parallel development. (UCM manages branches for you.) A *branch* is an object that specifies a linear sequence of versions of an element. Every element has one *main branch*, which represents the principal line of development, and may have multiple *subbranches*, each of which represents a separate line of development. As shown in

Figure 7, a project team can use many branches concurrently. In this example, the **main** branch represents new development work, an **alpha_port** subbranch is a port to a new platform, an **r1_bugs** subbranch contains fixes for bugs found in the first release, and so on.

Figure 7 Branching Hierarchy in Base ClearCase



To integrate work from one branch to another, you *merge* from the subbranch to the other branch. In the example above, assume that all work must be merged to the **main** branch before it can be included in a release for your product. After the fix on the **bug102** branch is tested and deemed ready for integration, you merge the work first to the **r1_bugs** branch. At some point, you test all the bug-fixing work on the **r1_bugs** branch, and when that work is ready to be incorporated into the main project, you merge from that branch to the **main** branch.

Of course, this is a very simple example of how you can define a branching strategy; the possibilities of branching and merging combinations are almost infinite.

Creating Standard Config Specs

As a project manager in an environment in which multiple branches are used, you must ensure that developers are working on the correct branches. To do that, you must ensure that the views they are using access and change the appropriate directory and file versions (that is, that they are accessing the appropriate branch).

The rules in the view's *config spec* determine which versions to select, and thus, which branch the developer is using. To ensure that developer views are configured properly, you can create a standard config spec and instruct all developers to use it.

Using ClearCase Metadata to Implement Development Policy

To enforce development policies in base ClearCase, a project manager or ClearCase administrator can create *metadata* to preserve information about the status of versions. To monitor the progress of the project, you can generate a variety of reports from this data and from the information that ClearCase captures in event records. ClearCase metadata you can use to define project policy includes:

- Version labels
- Attributes
- Hyperlinks
- Triggers
- Locks

See *Managing Software Projects with ClearCase* for details about using ClearCase metadata.

Using the ClearQuest-ClearCase Integration

If your project uses the ClearQuest-ClearCase integration, ClearQuest records can be associated with ClearCase versions, identifying which versions were created while making a particular change.

See the online documentation for the ClearQuest-ClearCase Integration Configuration program for details about this integration.

Developing and Building Software with ClearCase

4

This chapter provides an overview of how developers create, change, and build software in ClearCase, using either UCM or base ClearCase.

See *Developing Software with ClearCase* for details about the development process and *Building Software with ClearCase* for details about the build process.

4.1 Developing Software Using UCM

UCM structures the efforts of your software development team into a defined, repeatable process. This section provides an overview of the workflow for developers in UCM.

Joining a Project

A developer starts work by joining a UCM *project*. When you join a project, ClearCase creates two work areas: a development work area and an integration work area.

Development Work Area

A development work area allows you to work on your development activities in isolation from the project team. It consists of a development stream and a development view.

The *development stream* determines which versions of elements appear in your development view and maintains a list of your activities. You use the *development view* to access and change the files and directories in the components for your project.

Integration Work Area

An integration work area allows developers to test and deliver their work when they are ready to share it with the rest of the project team. The integration work area consists of an integration view associated with an integration stream to use for the project. There is only one integration stream per project, while there can be many development streams (typically, one per developer).

The integration stream determines which versions of elements appear in an integration view and maintains the project's baselines and activities that have been delivered from other development work areas.

Working on Activities

All work on your development stream takes place as part of a UCM *activity.* An activity is an object that tracks the work required to complete a development task.

Finding or Creating an Activity for Your Work

If your project uses Rational ClearQuest, you can use a to-do list in ClearQuest to access activities that you, your project manager, or other team members assign to you.

You can also create and use activities when you check out files and directories.

Modifying and Testing Source Files

To modify source files, go into your development view and check them out. When you want to keep a record of a file's current state, check it in. Any work you check in from your development view is not available to other team members until you deliver it.

Make sure the changes in your development view build and function properly before you deliver them.

Delivering Activities

When you are ready to make one or more of your activities available to the project team, you deliver them from your development stream to the project's integration stream.

Starting the Deliver Operation

When you start a deliver operation, ClearCase integrates the changes from your development work area to the integration work area. At this point, the files are checked out to your integration view.

Testing Your Work

You should build and test your work against the latest project work. To do this, use your integration view to access both the versions you delivered from your development work area and the latest versions delivered by the other developers working on the project.

Completing the Deliver Operation

When you are satisfied that your changes are compatible with the latest work for the project, you complete the deliver operation. (If you are not satisfied, you can cancel it.)

The deliver operation checks in the files that were integrated from the development work area to the integration work area at the beginning.

Delivering with MultiSite

If your project uses MultiSite to share source data with developers in other geographical locations, you may use a different method for delivering activities.

If a different site is responsible for controlling your project's source data, your organizational policy may require that you notify the integrator or project manager at that site when you deliver changes. That person merges your activities to the integration stream and tests your work.

Rebasing Your Work Area

Periodically, your project manager groups delivered activities into *baselines*, which are versions of each component in the project. Some of these baselines constitute a stable and significant

source configuration; your project manager will recommend that you rebase your development work area to the recommended configuration.

Starting the Rebase Operation

When you start the rebase operation, ClearCase integrates the versions specified by the recommended baseline in the project's integration stream into your development work area. At this point, the files are checked out to your development view.

Testing Your Development Work Area

You should test your work against the latest project work. To do this, use your development view to access both the versions you integrated from the integration stream and the latest (undelivered) versions in your development work area.

Completing the Rebase Operation

When you are satisfied that the recommended baseline is compatible with the work you have done in your development stream, you complete the rebase operation. (If you are not satisfied, you can cancel it.)

The rebase operation checks in the files that were integrated from the integration stream to the development work area at the beginning.

4.2 Developing Software Using Base ClearCase

This section provides an overview of the workflow for developers using base ClearCase functionality.

Setting Up a View

Typically, a project manager has defined the development policies for your project, and has implemented them using a configuration specification (or *config spec*).

To start working on a project, a developer creates a ClearCase view and then changes the config spec for that view to match the project's config spec.

Accessing and Modifying Files in Your View

To modify source files, go into the development view and check them out. When you want to keep a record of a file's current state, check it in.

Working on Branches

Typically, your project manager has defined a branching strategy for your project or organization, and has provided a standard config spec to ensure that developers are working on the branch appropriate for the project.

Using a Private Branch

Occasionally, you might want to isolate some short-term development effort from the project branch. For example, you may want to experiment with some changes to the product, but are not yet sure whether to include such experimental changes in official project builds. You can do this by creating a private branch based on the project branch. To do this, you change the rules in the config spec for your view.

If you decide that your changes should be incorporated into the project, you can then merge the changes on your private branch back to the project branch. If you decide to abandon your changes, you simply do not merge the work. In either case, you change your config spec rules back to the standard project config spec, and resume your work on the project branch.

Developing Software with ClearCase contains more information about creating private branches for development work and merging your work back to the project branch.

MultiSite Branch Mastership

If your organization uses ClearCase MultiSite to distribute development among multiple geographical sites, you may have to consider issues about branch control (mastership) between sites. See *ClearCase MultiSite Manual* for details.

4.3 Using ClearCase Build Tools

ClearCase supports *makefile*-based building of software systems, and provides a software build environment closely resembling that of the **make** program. **make** was developed for UNIX systems, and has been ported to other operating systems. You can use ClearCase-controlled files to build software, and use native **make** programs, third-party build utilities, your company's own build programs, or the ClearCase build tools **clearmake**, **omake**, and **clearaudit**.

The ClearCase build tools, **clearmake** and **omake**, provide compatibility with other **make** variants, along with powerful enhancements:

- *Build auditing*, with automatic detection of source dependencies, including header file dependencies
- Automatic creation of permanent bill-of-materials documentation of the build process and its results
- Sophisticated build-avoidance algorithms to guarantee correct results when building in a parallel development environment
- Sharing of binaries among views, saving both time and disk storage
- Parallel building, applying the resources of multiple processors and/or multiple hosts to builds of large software systems

The **clearaudit** build tool provides build auditing and creation of bill-of-materials documentation.

See *Building Software with ClearCase* for details about using ClearCase build tools.

5 Managing Software Projects with ClearCase

This chapter provides an overview of how project managers coordinate and track existing projects in ClearCase, using UCM and base ClearCase functionality.

See *Managing Software Projects with ClearCase* for detailed information about planning, creating, and managing software projects using ClearCase.

5.1 Managing Projects with UCM

This section summarizes the capabilities provided by UCM for managing software projects.

Adding Components to Projects

Over time, as project manager, you may need to add components to a project's integration stream. You do this by creating a new baseline that contains the new components. When the developers rebase their development work areas, the new components become visible.

Integrating MultiSite Development Work into the Project

In most cases, developers complete the deliver operations that they start. However, in a MultiSite configuration where the project's integration stream is mastered at a different replica than the developer's development stream, the developer cannot complete the deliver operations.

When ClearCase detects such a stream mastership situation, it makes the deliver operation a *remote deliver* operation. ClearCase starts the deliver operation but leaves it in the posted state. The project manager is responsible for finding and completing deliver operations in the posted state.

NOTE: Developers who have deliver operations in the posted state cannot rebase their development streams until the project manager completes or cancels their remote deliver operations.

See *ClearCase MultiSite Manual* for details about MultiSite mastership. See *Managing Software Projects with ClearCase* for details about using MultiSite with UCM, including finding and completing posted deliveries.

Managing Baselines

This section summarizes how project managers create, promote, and demote baselines.

Creating New Baselines

As developers deliver work to the integration stream, it is important that the project manager frequently make new baselines that incorporate the changes. Developers can then rebase to the new baselines and stay current with changes in the project.

Promoting and Demoting Baselines

As work on the project progresses, and the quality and stability of the components improve, the project manager can change a baseline's *promotion level* attribute to reflect the level of testing that the baseline has passed.

On occasion, a project manager may need to demote a baseline by changing its promotion level to one that is lower in the promotion level order. For example, suppose that after you create a new baseline, you discover that it contains a major bug. To prevent developers from introducing

this bug to their development streams by rebasing, you can demote the baseline to a Rejected level.

Tracking Projects

UCM provides several tools to help project managers track the progress of projects.

Comparing Baselines

ClearCase enables you to display the differences between UCM baselines graphically. You can compare baselines by the activities or versions in each baseline.

Using ClearQuest to Track Work

If you use the UCM-ClearQuest integration, developers and project managers can use ClearQuest queries, reports, and charts to retrieve information about the state of the project. For example:

- Which activities for a given project, stream, or developer are active
- Which activities are currently assigned to you (the to-do list)
- Detailed information for a particular activity, such as its state, owner, and changes made
- Trends in activity properties over time

You can also create custom ClearQuest queries, reports, and charts.

5.2 Managing Projects with Base ClearCase

This section summarizes the capabilities provided by base ClearCase for managing software projects.

Adding VOBs to Projects

Over time, you may need to add VOBs to a project's configuration. You should attach a ClearCase label to the initial versions in the VOB to represent the initial configuration of the VOB content.

Integrating Work Between Branches

As discussed in *Deciding a Branching and Merging Strategy* on page 15, base ClearCase functionality uses branches to isolate parallel development efforts. At some point, the project manager integrates the changes made on subbranches into a main product branch, which is sometimes called the integration branch.

In the simplest parallel development model, the main branch is the integration branch and a subbranch represents a separate development effort. Once the work on the subbranch is deemed ready for integration, the project manager merges the work from that branch to the main branch.

See *Managing Software Projects with ClearCase* for details about integrating parallel development using base ClearCase functionality.

Integrating MultiSite Development Work into the Project

In the standard MultiSite model, development at different sites occurs on branches of different types, and each site-specific branch type is mastered by the replica at that site. Integration merges occur only at the site whose replica masters the integration branch.

In a MultiSite configuration where the project's integration branch is mastered at a different replica than the developer's branch, you must manage integrations from the developer branch to the integration branch.

See *ClearCase MultiSite Manual* for details about managing projects and integrating development work in ClearCase MultiSite.

Glossary

ABE. (UNIX platforms only) See *audited build executor.*

ABSOLUTE VOB PATHNAME. (Windows platforms only) A pathname to a VOB object that begins with the *VOB-tag*. (That is, the pathname does not specify a network drive or view.) For example, **\myvob\src\test.c**, where the VOB-tag is **\myvob**.

ACCESS MODE. A three-digit octal number — 751, 644, or 777, for example. From left to right, the three digits designate file access permissions for three classes of user: *owner*, *group*, and others. The three bits of each of the three octal digits further define file access as read, write, or execute with respect to the three classes of users. The command **cleartool protect –chmod** controls an *element*'s access mode.

ACTIVATE. Make a *dynamic view* or VOB accessible on a particular host. See the **mount** reference page. On UNIX platforms, see also the **setview** reference page.

ACTIVE. A *VOB* becomes active on a host when it is mounted with the **cleartool mount** command.

A *dynamic view* becomes active on a host when it is started with either a **cleartool setview** or **startview** command on UNIX platform; or a **Connect Network Drive**, **net use**, or **startview** command on Windows platforms. These commands establish a connection between the host's MVFS file system and the dynamic view's *view_server* process.

ACTIVITY. A ClearCase UCM object that tracks the work required to complete a development task. An activity includes a text headline, which describes the task, and a change set, which identifies all versions that you create or modify while working on the activity. When you work on a version, you must associate that version with an activity. If your project is configured to use the UCM-ClearQuest integration, a corresponding ClearQuest record stores additional activity information, such as the state and owner of the activity.

ADMINISTRATIVE VOB. A VOB containing global type objects, which are copied to client VOBs on an as-needed basis when users wish to create instances of the type objects in the client VOBs. See *auto-make-type*, *global type*, *local copy.*

ADMINISTRATOR. (Windows platforms only) The *Administrator* user, or any member of the *Administrators* group. Some operations, such as stopping and starting ClearCase, require

membership in the *Administrators* group only on the client workstation. Other operations, such as modifying domain-wide group assignments, require membership in the *Administrators* group on the domain controller for the *Windows NT Server domain*.

ALBD_SERVER. Atria Location Broker Daemon. This ClearCase master server runs on each ClearCase host; it starts up, and dispatches messages to, the various ClearCase server programs (for example, **view_server**, **vob_server**, **db_server**, **vobrpc_server**, and so on) as necessary. See the **albd_server** reference page.

ALL-ELEMENT TRIGGER TYPE. A trigger type that is automatically associated with all elements in a VOB.

ANCESTOR. In an element's version tree, a version that is on the line of descent of another version. In other words, a version that has contributed to the contents of another version is considered an ancestor to the latter version.

ANNOTATION BOX. (UNIX platforms only) Part of the **xcleardiff** display, showing how lines of one file differ from lines of other files, with which it is being compared or merged.

ARGUMENT. A word or quoted set of words processed by a command.

ATRIA LOCATION BROKER. See *albd_server*.

ATTACHE CLIENT. (Attache) The program through which you enter all commands, using the *command-line* and graphical interfaces. This program, **attache.exe**, is located in the **bin** subdirectory within the Attache installation area on your PC (*attache-home-dir***bin****attache.exe**).

ATTACHE WINDOW. (Attache) The standard Microsoft Windows window in which the *Attache client* program executes.

ATTACHED LIST. See *trigger inheritance*.

ATTRIBUTE. A *meta-data* annotation attached to an *object*, in the form of a name/value pair. Names of attributes are specified by user-defined *attribute types*; values of these attributes can be set by users. Example: a project administrator creates an attribute type whose name is **QAed**. A user then attaches the attribute **QAed** with the value `"Yes"` to versions of several file elements.

ATTRIBUTE TYPE. An *object* that defines an attribute name for use within a *VOB*. It constrains the attribute values that can be paired with the attribute name (for example, an integer in the range 1–10).

ATTRIBUTE VALUE. See *attribute type*.

AUDIT, AUDITED SHELL. See *build audit*.

AUDITED BUILD EXECUTOR. (UNIX platforms only) A process invoked through the UNIX remote-shell facility, in order to execute one or more build scripts on behalf of a remote **clearmake**.

AUTO-MAKE-BRANCH. ClearCase's facility, specified in a config spec rule, for automatically creating one or more branches when a **checkout** is performed.

AUTO-MAKE-TYPE. ClearCase's facility for automatically copying *type objects* from an *administrative VOB* to a client VOB, when a user attempts to make an instance of the type object in the client VOB. See *global type*, *local copy*.

AUTOMOUNTED. A file system on a remote host that has been automatically mounted on the local host.

BACKUP REGISTRY SERVER HOST. See *registry server host*.

BASE CONTRIBUTOR. In ClearCase comparison and merge tools, the *contributor* against which all other contributors are compared when reporting differences. For example, if you are comparing four contributors, a through d, and contributor a is the base contributor, ClearCase compares:

- Contributor a against contributor b.
- Contributor a against contributor c.
- Contributor a against contributor d.

BASELEVEL. A matched set of source versions, representing a certain milestone in a project. Typically, a baselevel is recorded by attaching a version label to each version in the set.

BASELEVEL-PLUS-CHANGES. A software development methodology, in which the current status is determined in relation to a recent baselevel.

BASELINE. A ClearCase UCM object that typically represents a stable configuration for one or more components. A baseline identifies activities and one version of every element visible in one or more components. You can create a *development stream* or *rebase* an existing development stream from a baseline.

BIDIRECTIONAL. See *hyperlink*.

BITMAP FILES. Files that store bitmaps for the icons displayed by ClearCase GUI programs.

BOS FILE. A file containing rules that specify settings of *make macros*, which affect the way in which a *target rebuild* proceeds.

BRANCH. An *object* that specifies a linear sequence of *versions* of an *element*. The entire set of versions of an element is called a *version tree*; it always has a single *main branch*, and may also have *subbranches*. Each branch is an instance of a *branch type* object.

BRANCH NAME. See *branch type*.

BRANCH PATHNAME. A sequence of branch names, starting with **main** (the name of an element's starting branch). Examples: **/main/motif** on UNIX platforms, or **\main\maintenance\bug459** on Windows platforms.

BRANCH TYPE. An *object* that defines a *branch name* for use within a *VOB*.

BROADCAST MESSAGE SERVER. (UNIX platforms only) A Hewlett-Packard SoftBench program that passes messages among SoftBench applications.

BROWSER. A class of ClearCase graphical tools. The various kinds of browser —type object, vtree, properties, history, log, VOB admin, view-tag, VOB-tag, pool, list, and so on— share a common model: they display some kind of ClearCase data, and they include toolbar and

menu commands for operating on that data. In addition, as you perform your ClearCase work, browsers prompt you automatically for any data required to complete particular ClearCase operations.

BROWSER WINDOW. (Attache) When the Attache window is divided into upper and lower partitions, the upper partition is the browser window. The *File Browser* is displayed in the browser window.

BUILD. The process during which a ClearCase build program (**clearmake**, **clearaudit**, or **omake**) produces one or more *derived objects*. (**NOTE**: **omake** is supported only on Windows platforms.) This may involve actual translation of source files and construction of binary files by compilers, linkers, text formatters, and so on. A system build consists of a combination of actual *target rebuilds* and *build avoidance*. See also *express build*.

BUILD AUDIT. The process of recording which files and directories (and which versions of them) are read or written by the operating system during the execution of one or more programs. A client host's MVFS file system performs an audit during execution of a ClearCase build program: **clearmake**, **omake**, **clearaudit**, or **abe**. (**NOTE**: **omake** is supported only on Windows platforms, and **abe** is supported only on UNIX platforms.) When the build audit ends, the build program creates one or more *configuration records* (CRs). An *audited shell* is a UNIX shell process created by **clearaudit** in which all file system accesses are audited, and are recorded in a *configuration record* when the shell exits.

BUILD AVOIDANCE. The ability of a ClearCase build program to fulfill a build request by using an existing derived object, instead of creating a new derived object by executing a build script. The build program can *reuse* a derived object currently in the view or *wink in* a derived object that exists in another view. The process by which the build program decides how to produce a derived object is called *configuration lookup*.

BUILD CONFIGURATION. The set of source versions in a view, the current build script that would be executed, and the current build options. See also *configuration lookup*.

BUILD DEPENDENCY. See *dependency*.

BUILD HOSTS FILE. (UNIX platforms only) A file that lists hosts to be used in a parallel build.

BUILD OPTIONS SPECIFICATION FILE. See *BOS file*.

BUILD REFERENCE TIME. The time at which a *build session* (invocation of a ClearCase build program) begins. Versions created after this time are kept out of the build.

BUILD SCRIPT. The set of shell commands that a ClearCase build program or a standard **make** program reads from a *makefile* when building a particular *target*.

BUILD SERVER CONTROL FILE. (UNIX platforms only) A file on a build host that controls its availability as a build server.

BUILD SERVER HOST. (UNIX platforms only) A host used to execute build scripts during a **clearmake** parallel build.

BUILD SESSION. A top-level invocation of a ClearCase build program; during the session, recursive invocations of **clearmake**, **omake**, or **clearaudit** may start subsessions. (**NOTE**: **omake** is supported only on Windows platforms.)

BUILD TARGET. A word, typically the name of an object module or program, that can be used as an argument in a **clearmake** or **omake** command. (**NOTE**: **omake** is supported only on Windows platforms.) The target must appear in a *makefile*, where it is associated with one or more *build scripts*.

BUILT-IN RULES. Build rules defined in a system-supplied or ClearCase-supplied file, which supplement the explicit build rules in a user's *makefiles*.

BUMP. The taking away of a ClearCase license from a lower-priority user by a higher-priority user.

CANDIDATE. A *derived object* that is being considered for *winkin* or reuse during *configuration lookup*.

CASCADING MENU. A menu that includes one or more sub-menus.

CATALOGED. Names of elements and VOB symbolic links that appear in a version of a directory element are said to be *cataloged* in the directory version. A *derived object* is said to be cataloged (and, hence, available for *reuse* and *winkin*) in a particular VOB.

CHANGE SET.

A list of related versions associated with a UCM *activity*. ClearCase records the versions that you create while you work on an activity. An activity uses a change set to record the versions of files that are delivered, integrated, and released together.

CHECKED-OUT VERSION. A placeholder object in a VOB database, created by the **checkout** command. This object corresponds to the view-private object (file or directory) that you work with after checking out the element.

CHECKING (A VOB). Finding and fixing inconsistencies between a VOB database and the VOBs *storage pools*. See *VOB database snapshot* and the **checkvob** reference page.

CHECKOUT/CHECKIN. The two-part process that extends a *branch* of an *element*'s *version tree* with a new *version*. The first part of the process, *checkout*, expresses your intent to create a new version at the current end of a particular branch. (This is sometimes called checking out a branch.) The second part, *checkin*, completes the process by creating the new version.

For *file elements*, the checkout process creates an editable version of the file in the *view* (and, with Attache, in your *workspace*), with the same contents as the version at the end of the *branch*. Typically, a user edits this file, then checks it back in.

For *directory elements*, the checkout process allows file elements, (sub)directory elements, and VOB symbolic links to be created, renamed, moved, and deleted.

Performing a checkout of a branch does not necessarily guarantee you the right to perform a subsequent checkin. Many users can checkout the same branch, as long as they are working in different views. At most one of these can be a *reserved checkout*, which guarantees the user's right to checkin a new version. An *unreserved checkout* affords no such guarantee. If several users have unreserved checkouts on the same branch in different views, the first user to perform a checkin wins — another user must perform a *merge* if he wishes to save his or her checked-out version.

CHECKOUT RECORD. The event record created by the **checkout** command.

CLEARCASE GROUP. (Windows platforms only) A special group, usually created in the Windows NT domain when ClearCase is installed. Only ClearCase administrative accounts and the login account for the ClearCase ALBD Service should be members of this group.

CLEARCASE REGISTRY. A set of files on the *registry server host* that map logical VOB and view names (*VOB-tags* and *view-tags*) to physical storage locations (*VOB storage directories* and *view storage directories*). See **registry_ccase**, *backup registry servers*.

CLEARTEXT FILE. An ASCII text file that contains a whole copy of some version of an element, having been extracted from a *data container* that is in compressed format or *delta* format. A ClearCase *type manager* creates a cleartext container the first time it accesses the version. Subsequent reads of that version access the cleartext file, for better performance.

CLEARTEXT POOL. A VOB *storage pool*, used for *data containers* that contain *cleartext*.

CLI. Command-line interface.

CLIENT. The programs invoked by users: **cleartool**, **clearmake**, **cleardiff**, and other programs located in the ClearCase **bin** directory. See *Attache client*.

CLOCK SKEW. The discrepancies among the system clocks of several hosts.

COMMAND OPTION. In the command-line interface (CLI), a word beginning with a hyphen (–) that affects the meaning of a command.

COMMAND PROMPT. (Attache) The character string (the name of the *current working directory*, followed by ">") that Attache displays when it is ready to execute a command.

COMMAND WINDOW. (Attache) The part of the Attache window where you type commands, and where command output is displayed. When the Attache window is divided into upper and lower partitions, the lower partition is the command window.

COMMAND-LINE INTERFACE. The input method in which you type a command at a *command prompt*.

COMMENT DEFAULT. The action taken by certain CLI commands when you do not specify a comment-related option.

COMMON ANCESTOR. In an element's version tree, a version that is on the line of descent of two (or more) versions on different branches.

COMPATIBILITY MODE. A **clearmake** or **omake** execution mode, in which it emulates another **make** variant. (**NOTE**: **omake** is supported only on Windows platforms.)

COMPONENT. A ClearCase object that you use to group a set of related directory and file elements within a UCM *project*. Typically, you develop, integrate, and release the elements that make up a component together. A project must contain at least one component, and it can contain multiple components. Projects can share components.

COMPRESSED_FILE. The ClearCase element type that uses data compression on individual versions.

COMPRESSED_TEXT_FILE. The ClearCase element type that uses both *delta* management and data compression on individual versions.

CONFIG SPEC. A set of configuration rules specifying which versions of VOB elements a view selects. The config spec for a *snapshot view* also specifies which elements to load into the view.

See *pattern*, *scope*, *version-selector*, *version-selection rule*, *load rule*, and the **config_spec** reference page.

CONFIGURATION (OF A DERIVED OBJECT). The bill-of-materials information recorded in a derived object's configuration record, including versions of source files used to build the object, build script, and build options.

CONFIGURATION (OF A VIEW). The set of *versions* (one version of each *element*) selected by a view's *config spec*.

CONFIGURATION LOOKUP. The process by which a ClearCase build program determines whether to perform a *target rebuild* of a derived object (execute a *build script*) or reuse an existing instance of the derived object. This involves comparing the *configuration records* of existing derived objects with the *build configuration* of the current view.

CONFIGURATION MANAGEMENT. The discipline of tracking the individual objects and collections of objects (and the versions thereof) that are used to build systems.

CONFIGURATION RECORD (CR). A listing produced by a *target rebuild*, logically associated with each *derived object* created during the rebuild. A configuration record is a bill of materials for a derived object, indicating exactly which file system objects (and which specific versions of those objects) were used by the rebuild as input data or as executable programs, and which files were created as output. It also contains other aspects of the *build configuration*.

Each *target rebuild* typically involves the execution of a single build script, and creates a single configuration record. If a target has *subtargets* that must be rebuilt, also, a separate configuration record is created for each subtarget rebuild.

CONFIGURATION RECORD HIERARCHY. A tree structure of configuration records, which mirrors the hierarchical structure of targets in the *makefile*.

CONFIGURATION RULE. See *config spec*.

CONFIGURATION SPECIFICATION. See *config spec*.

CONTAINER. See *data container*.

CONTEXT. See *view context*.

CONTRIBUTOR. A file, directory, or *version* considered for a comparison or merge. Typically, contributors are multiple versions of the same ClearCase file or directory element.

CONVERSION SPECIFICATION. A code (for example, **%En**) that is part of the *format string* specification following a **–fmt** option. See the **fmt_ccase** reference page.

CR. See *configuration record*.

CROSS-VOB HYPERLINK. A *hyperlink* that connects two objects in different VOBs. The hyperlink always appears in a **describe** listing of the *from* object. It also appears in a listing of the *to* object, unless it was created as a *unidirectional* hyperlink (**mkhlink –unidir**). See *same-VOB hyperlink*.

CURRENT WORKING DIRECTORY. (ClearCase) The context in which *relative pathnames* are resolved by the operating system. This can be a location in ClearCase's *extended namespace*.

(Attache) The directory within your *workspace* (and the corresponding directory within some VOB) with respect to which the *command-line interface* interprets simple file names and relative pathnames of ClearCase data.

CURRENT WORKSPACE. (Attache) The *workspace* that is active in the Attache window. The *workspace name* appears in the *title bar*.

DATA CONTAINER. A file (or directory) that contains the data produced by a *build script*. A data container and a *configuration record* are the essential constituents of a *derived object*. Also, a file in a *source pool* or *cleartext pool*, containing the data for one or more *versions* of a file *element*.

DEFAULT CONFIG SPEC. See *config spec*.

DEGENERATE DERIVED OBJECT. A *derived object* that cannot be successfully processed, because its *data container* and/or associated *configuration record* are not available.

DELIVER. A ClearCase operation that allows developers to share their work with the rest of the project team by merging work from their own *development streams* to the project's *integration stream*. If required, the deliver operation invokes the Merge Manager to merge versions.

DELTA. The incremental difference (or set of differences) between two versions of a file *element*. Certain type managers (for example, **text_file_delta**), store all versions of an element in a single *data container*, as a series of deltas.

DEPENDENCY. In a *makefile*, a word listed after the colon (:) on the same line as a *target*. A *source dependency* of a target is a file whose version-ID is taken into account in a configuration lookup of the target. A *build dependency* is a derived object that must be built before the target is built.

DERIVED OBJECT (DO). An MVFS file (pathname within a VOB) produced by a **clearmake** or **omake** build or a **clearaudit** session. (**NOTE**: **omake** is supported only on Windows platforms.) Each derived object is associated with the *configuration record* that is created by the ClearCase build program to document the build. A shareable DO can be winked in by other views. A nonshareable DO cannot be winked in by other views unless you explicitly make it available.

DERIVED OBJECT SCRUBBING. The removal of data containers from view storage, of data containers from derived object pools in a VOB, and of derived objects themselves from a VOB database.

DERIVED OBJECT SHARING. Multiple views simultaneously using the same *derived object*. See *winkin*.

DERIVED OBJECT STORAGE POOL. A *storage pool* for the *data containers* of a VOB's *derived objects*. Only those derived objects that have been winked in are stored in these pools. Data containers of unshared and nonshareable derived objects are stored in view-private storage.

The first time a derived object is *winked in*, the **promote_server** program copies the data from the original view, creating a data container in a derived object storage pool.

DERIVED OBJECT VERSION. See *DO version*.

DETECTED DEPENDENCY. A source *dependency* that is automatically detected by **clearmake** or **omake**, rather than being explicitly coded in a *makefile*. (**NOTE**: **omake** is supported only on Windows platforms.)

DEVELOPMENT STREAM. A ClearCase UCM object that determines which versions of elements appear in your *development view*, and maintains a list of your activities. The purpose of the development stream is to let you work on a set of activities and corresponding versions in isolation from the rest of the project team. The development stream configures your development view to select the versions associated with the foundation baselines plus any activities and versions that you create after joining the project or rebasing your development stream.

DEVELOPMENT VIEW. A view associated with a UCM *development stream*.

Use a development view to work on a set of *activities* and corresponding *versions* isolated from the rest of the project team. You then can share changes made in a development view with the rest of the project team by *delivering* activities to the project's *integration stream*.

A development view can be either a *dynamic view* or a *snapshot view.*

DIRECT MODE. (UNIX platforms only) A state of the *hyperlink tree browser* in which the browser displays only the *hyperlinks* that are directly connected to the displayed objects. Compare this to the hyperlink tree browser's *inheritance mode*.

DIFFERENCE. (When comparing files, directories, or versions, a section where the content of the *base contributor* is unlike that of one or more contributors.

DIRECTORY ELEMENT. An *element* whose versions are like directories — they catalog the names of *file elements*, other *directory elements*, and *VOB symbolic links*.

DIRECTORY VERSION. A version of a *directory element*.

DISTRIBUTED BUILD. (UNIX platforms only) A *parallel build* in which execution takes place on multiple hosts in a local area network.

DO. See *derived object*.

DO-ID. A unique identifier for a derived object, including a time stamp and a numeric suffix to guarantee uniqueness. Example: the substring beginning with **@@** in **hello.o@@12-May.19:15.232**.

DOMAIN. (Windows platforms only) See *Windows NT Server domain*.

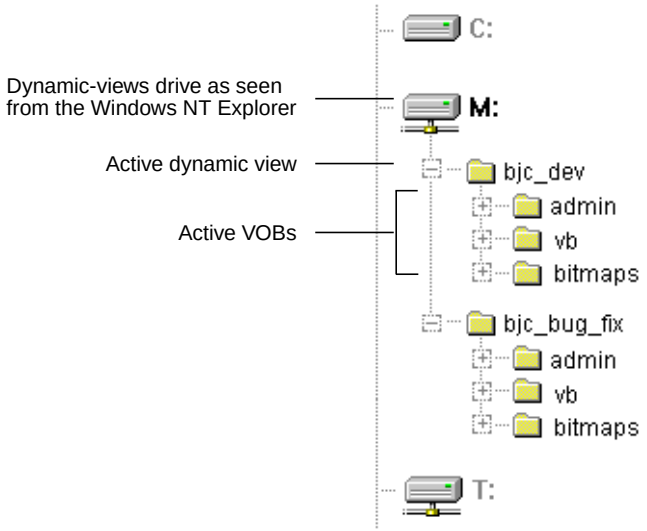
DO VERSION. A derived object that has been checked in as a version of an element.

DOWNLOADING FILES. (Attache) Copying files **to** your *current workspace* **from** its associated *view.*

DRIVE-RELATIVE PATHNAME. (Windows platforms only) A full pathname that does not include a drive specification (for example, **\project\src**).

DYNAMIC VIEW. A *view* that is always current with the VOB (as specified by the *config spec*). Dynamic views use the *MVFS* to create and maintain a directory tree that contains *versions* of VOB *elements* and *view-private files*. Dynamic views are not supported on all ClearCase platforms.

DYNAMIC-VIEWS DRIVE. (Windows platforms only) A drive, **M:** by default, that provides access to the VOBs and dynamic views *active* on the current ClearCase host.



DYNAMIC-VIEWS ROOT DIRECTORY. The directory maintained by the *MVFS* file system in which *view-tag* entries appear, allowing views to be accessed. On Windows, this directory is **\\view** or **M:**\; on UNIX, it is **/view**. See also *viewroot directory*.

ECLIPSED. Invisible, because another object with the same name is currently selected by the current view.

ELEMENT. An *object* that encompasses a set of *versions*, organized into a *version tree*.

ELEMENT TRIGGER TYPE. See *trigger type*.

ELEMENT TYPE. A class of versioned file or directory objects. ClearCase supports predefined element types (for example, **file**, **text_file**). Users can define additional types (for example, **c_source_file**) that are refinements of the predefined types. When an *element* is created, it is assigned one of the currently-defined element types in its VOB. Each user-defined element type is implemented as a separate VOB object.

ELLIPSIS. The *wildcard* symbol "...". In a *version-selector*, it indicates zero or more directory levels.

EMULATION MODE (OMAKE). (Windows platforms only) **omake**'s execution mode in which its behavior resembles that of other build utilities, such as PolyMake. See also *compatibility mode*.

ENABLED. An *enabled* button or menu command is active and can be activated with a mouse click. Opposite: *insensitive*.

ENCAPSULATOR. A program that packages the functionality of an external software system.

EPOCH NUMBER. (MultiSite) An integer associated with a ClearCase operation performed at a replica. Each replica records the epoch numbers of operations it has performed and of operations it has received from other replicas.

EPOCH NUMBER MATRIX. (MultiSite) A complete set of epoch numbers, indicating the local VOB replica's estimate of the current state of all replicas in a *VOB family*. A replica's own epoch row within the matrix reflects its actual state.

EVENT. A ClearCase operation that is recorded by an *event record* in a VOB's event *history*.

EVENT-ID. A numeric identifier, which can be used to specify a particular event record in the **chevent** subcommand.

EVENT RECORD. An item in a VOB database that contains information about an operation that modified that VOB.

EXCEPTION LIST. The set of users to whom a *lock* or *trigger* will not apply.

EXPIRATION PERIOD. (MultiSite) The interval after which the *store-and-forward* facility stops trying to process a *shipping order*.

EXPORT VIEW. (UNIX platforms only) A view used to export a VOB to a non-ClearCase host.

EXPRESS BUILD. A build during which the ClearCase build program creates derived objects, but does not write information about them into the VOB. This speeds up the build and means that the DOs cannot be winked in by other views.

EXTENDED NAMESPACE. ClearCase's extension of the standard Windows or UNIX pathname hierarchy. Each host has a *view-extended namespace*, allowing a pathname to access VOB data using any view that is active on that host. Each VOB has a *VOB-extended namespace*, allowing a pathname to access any version of any element, independently of (and overriding) version-selection by views.*Derived objects* also have extended pathnames, which include *DO-IDs*. See *namespace*.

EXTENDED NAMING SYMBOL. A symbol (by default, **@@**) appended to an element name or derived object name, signaling the MVFS file system to bypass automatic version-selection by a view.

EXTENDED PATHNAME. A VOB-extended pathname specifies a particular location in an element's *version tree*, or a particular derived object cataloged in that VOB.

If the pathname specifies a particular version, is termed a *version-extended pathname*. Examples:

Windows:

```
foo.c@@\main\17
\myproduct\bin\xtract@@\REL_1
\myproduct@@\main\bug403\5
```

UNIX:

```
foo.c@@/main/17
/usr/myproduct/bin/xtract@@/RELEASE_1
/usr/myproduct@@/main/bug403/5
```

A *view-extended pathname* accesses a file system object in the context of a specified view. For an element, such a pathname specifies the version of the element selected by that view's *config spec*; for a view-private file or derived object, such a pathname accesses an object in the view's private storage area. Examples:

Windows (assuming that **M:** is the dynamic views root directory):

```
M:\akp\proj_1\foo.c
M:\archive\proj_1\foo
M:\bugfix\proj_1\to_do.list
```

UNIX:

```
/view/akp/usr/project/foo.c
/view/archive/usr/project/foo
/view/bugfix/usr/project/to_do.list
```

FAT FILE. (Windows platforms only) A file located in a file system of type FAT (file allocation table).

FEATURE LEVEL. An integer that Rational increments at each ClearCase release that introduces features that affect compatibility across VOB replicas running earlier ClearCase releases.

FIELD-WIDTH SPECIFIER. An optional part of a *conversion specification*, which helps in creating reports with fixed-width columns. See the **fmt_ccase** reference page.

FILE BROWSER. (Attache) A directory tree browser that you can display in the upper partition of the *Attache window.* The left side displays an icon for each of your workspaces, and initially, the *View* and *Workspace Contents* for the current workspace.

FILE CONTENTS. See *file system data*.

FILE ELEMENT. See *element*.

FILE SYSTEM CONFIGURATION. The set of *versions* accessible through a view.

FILE SYSTEM DATA. The bytes stored in a *version* of a *file element*. A file's contents are distinguished from its *meta-data* (*attributes*, *hyperlinks*, and so on).

FILE TYPE. The identifier returned by ClearCase file typing subsystem, through a lookup in ClearCase-supplied and/or user-supplied *magic files*. File types are used to select an *element type* for a new element.

FILE TYPING RULE. See *magic file*, *file type*.

FILENAME PATTERN. See *pattern*.

FIRE A TRIGGER. The process by which ClearCase verifies that the conditions defined in a *trigger* are satisfied, and causes the associated trigger action(s) to be performed.

FLAG FILE. A file whose existence/non-existence controls conditional processing in a *build script*.

FLAT. A non-hierarchical listing, combining information from a collection of *configuration records*.

FLAT FILE. A file that contains text characters only.

FORMAT STRING. In several **cleartool** subcommands, a character string argument to the **–fmt** option. This string can combine literal text with *conversion specifications*, to specify how *event record* data is to be formatted in the command's output. See the **fmt_ccase** reference page.

FOUNDATION BASELINE. A property of a stream. Foundation baselines specify the versions and activities that appear in your view. As part of a *rebase* operation, foundation baselines of the target stream are replaced with the set of recommended baselines from the source stream.

FROM-OBJECT. See *hyperlink*.

FROM TEXT. A string-valued *attribute* attached to a *hyperlink* object, conceptually at its *from* end.

FULL BASELINE. A *baseline* created by recording all versions below the component's root directory. Generally, full baselines take longer to create than *incremental baselines*; however, ClearCase can look up the contents of a full baseline faster than it can look up the contents of an incremental baseline.

FULL LOCAL PATHNAME. (Windows platforms only) A *full pathname* that includes a drive specification (for example, **D:\project\src**).

FULL PATHNAME. A standard operating system pathname beginning with a "/" on UNIX or a drive letter followed by "\" on Windows (for example, **C:** or **Z:**).

FULL REMOTE PATHNAME. A *full pathname* indicating a location in some ClearCase VOB.

G-FILE. (UNIX platforms only) A file produced by the SCCS **get** command.

GENERATED COMMENT. A comment string for an *event record* that is created automatically by **cleartool** — for example, the **checkin** comment for a new directory version.

GET. (Attache) The command that *downloads* one or more files to your *workspace*.

GLOBAL PATHNAME. A network-wide pathname for a ClearCase *view storage directory* or *VOB storage directory*. Some global pathnames are valid only within a particular *network region*.

GLOBAL TYPE OBJECT. A *type object*, created with **mk***xx***type –global** and located in an *administrative VOB*. Such objects are used by the *auto-make-type* facility to create *local copies* in other VOBs (termed *client VOBs*). See also: *administrative VOB*.

GNU MAKE. A **make** variant distributed by the Free Software Foundation.

GOAL TARGET. The target(s) explicitly named on a **clearmake** or **omake** command line. (**NOTE**: **omake** is supported only on Windows platforms.)

GROUP LIST. The list of groups to which your user account belongs (in addition to the Windows primary group name or UNIX principal group ID defined for your user account). VOBs, views, and file system objects also have group assignments, and the interaction of these assignments with your group memberships helps determine the outcome of VOB and view data storage access attempts.

HARD LINK. (UNIX platforms only) An additional name for a file system object, cataloged in the same directory or in a different directory. UNIX hard links are cataloged in standard UNIX directories; VOB *hard links* are cataloged in versions of *directory elements*.

HELPER HOST. (Attache) See *workspace helper host*.

HIJACKED FILE. A version in a *snapshot view* that is modified but not checked out. By default, a non-checked out version in a snapshot view is given the file attribute of **read-only**. If you change this attribute and modify the file, you have hijacked the file by taking it out of direct ClearCase control.

HISTORY. Meta-data in a *VOB*, consisting of *event records* involving that VOB's *objects*. The history of a file element includes the creation event of the element itself, the creation event of each version of the file, the creation event of each branch, the assignment of attributes to the

element and/or its versions, the attaching of hyperlinks to the element and/or its versions, and so on.

HISTORY BROWSER. A graphical tool that enables you to examine ClearCase *event records* for one or more objects.

HISTORY MODE. The state of a process whose current working directory is in the ClearCase *VOB-extended namespace* (for example, **hello.c@@/main** on UNIX or **hello.c@@\main** on Windows).

HOST-LOCAL PATHNAME. For a *view storage directory* or *VOB storage directory*, a pathname that specifies the directory's location in its own host's file system. This pathname need not be valid on any other host. Contrast this with the directory's *global pathname*.

HYBRID COMMAND. (Attache) A command that affects both your *workspace* (on your PC) and ClearCase storage (*view* and/or *VOB* located on one or more remote hosts).

HYPERLINK. A logical pointer between two objects. A hyperlink is implemented as a VOB object; it derives its name by referencing another VOB object, a *hyperlink type*. A hyperlink can have a from-string and/or to-string, which are implemented as string-valued attributes on the hyperlink object.

A bidirectional hyperlink connects two objects in the same VOB or in different VOBs, and can be navigated in either direction: *from-object* to *to-object* or *to-object* to *from-object*. A unidirectional hyperlink connects two objects in different VOBs, and can be navigated only in the *from-object* to *to-object* direction.

HYPERLINK TREE BROWSER. (UNIX platforms only) A graphical tool that enables you to traverse hyperlinks connecting objects.

HYPERLINK-ID. A system-generated identifier that, in conjunction with the name of the *hyperlink type*, uniquely identifies a *hyperlink* object. Examples: **@391@/usr/hw** on UNIX or **@391@\hw_vob** on Windows.

HYPERLINK INHERITANCE. The ClearCase feature by which hyperlinks attached to a version can be detected in a search for hyperlinks on a successor version.

HYPERLINK SELECTOR. A string that specifies a particular hyperlink. It consists of the name of a hyperlink type object, followed by a (possibly abbreviated) hyperlink-ID. Examples: **DesignFor@391@/usr/hw** on UNIX and **DesignFor@391@\hw_vob** on Windows.

HYPERLINK TYPE. An *object* that defines a hyperlink name for use within a *VOB*.

ICON. A small picture used by graphical tools.

ICON FILE. A file containing rules that map ClearCase *file types* to names of *bitmap files*.

INCREMENTAL BASELINE. A *baseline* created by recording the last *full baseline* and those versions that have changed since the last full baseline was created. Generally, incremental baselines are faster to create than full baselines; however, ClearCase can look up the contents of a full baseline faster than it can look up the contents of an incremental baseline.

INCLUSION LIST. A list of *type objects*, defining the scope of a *trigger type*.

INDIRECT FILE. (Attache) A *local file* containing a list of files to be used by the *get*, *put*, **find**, or **findmerge** commands.

INHERIT, INHERITANCE LIST. See *trigger inheritance*.

INHERITANCE MODE. (UNIX platforms only) A state of the *hyperlink tree browser* in which the browser displays both the *hyperlinks* that are directly connected to the displayed objects and the hyperlinks that are connected to ancestors of the displayed objects.Compare this to the hyperlink tree browser's *direct mode*.You can switch back and forth between these modes using the browser's menu.

INSENSITIVE. An *insensitive* button or menu command is greyed out and does not respond to pointer actions. Opposite: *enabled*.

INSTALLATION HOST. A host on which ClearCase has been (or is about to be) installed.

INSTALLATION METHOD. An installation-time choice, specifying which *installation host*(s) are to be addressed, and whether software is to be installed or deinstalled.

INSTALLATION MODEL. An installation-time choice, specifying how the software for a ClearCase family product is to be transferred from the network-wide *release area* to one or more *installation hosts*.

INSTANCE. See *type object*.

INTEGRATION BRANCH. A *branch* containing versions available to all members of a team. A team member often works in isolation on a *private branch*. To make private work available to others, the team member merges versions on the private branch with versions on the integration branch.

INTEGRATION STREAM. A ClearCase UCM object that enables access to versions of the project's shared elements. A *project* contains only one integration stream. The integration stream maintains the project's baselines. The integration stream configures integration views to select the versions associated with the *foundation baselines* plus any activities and versions that have been delivered to the integration stream.

INTEGRATION VIEW. A *view* associated with a UCM *project*'s *integration stream*.

Use an integration view to build and test the latest *versions* of a project's shared *elements*.

An integration view can be either a *dynamic view* or a *snapshot view.*

INTERACTIVE MODE. The mode of **cleartool** usage in which the program prompts you (with `cleartool>`) to enter a command, executes the command, then prompts you to enter another one. See *single-command mode*.

LABEL. See *version label*.

LABEL TYPE. A *type object* that defines a version label for use within a VOB.

LEAF NAME. The simple file or directory name at the end of a multiple-component pathname.

LICENSE. Permission for one user to run ClearCase programs and/or use ClearCase data, using any number of hosts in the local area network.

LICENSE DATABASE FILE. (UNIX platforms only) A file that defines a set of ClearCase licenses.

LICENSE DATABASE. (Windows platforms only) A data structure in the Windows NT Registry that defines a set of ClearCase licenses.

LICENSE PRIORITY. A slot in the scheme by which some ClearCase users can *bump* others, taking their licenses.

LICENSE SERVER HOST. A host whose **albd_server** process controls access to the licenses defined in its *license database file*.

LINE OF DESCENT. The sequence of element versions leading to a particular version. This sequence may traverse branches.

LINK TEXT. The text string that is the contents of a *VOB symbolic link*. (**NOTE**: On UNIX platforms, the link text can be the contents of a UNIX-level symbolic link.)

LOAD. Copy a version of an element to a snapshot view and keep track of the checkins, updates, and other ClearCase operations affecting the element.

LOAD RULE. A statement in the *config spec* that specifies an element or subtree to *load* into a *snapshot view*. Config specs can have more than one load rule. See also *version-selection rule*.

LOCAL COMMAND. (Attache) A command that affects only your *workspace* — not any ClearCase storage (*view* or *VOB*).

LOCAL COPY. During an *auto-make-type* operation, the copy of a *global type* object (located in an *administrative VOB*) that is created in a client VOB.

LOCAL FILE. (Attache) A file whose physical location is on your PC.

LOCAL SHELL. (Attache) A command shell executing on your PC.

LOCK. A mechanism that prevents a VOB object from being modified (for file system objects) or from being instanced (for *type* objects). See *obsolete*.

LOCK MANAGER. A ClearCase server that arbitrates transaction requests to all VOB databases on the local host. A **lockmgr** runs on each ClearCase host. (**NOTE**: On Windows NT hosts, the lockmgr runs as a Windows NT service.)

LOG BROWSER. A graphical tool that enables you to examine ClearCase log entries on one or more hosts in the network.

LOGICAL OPERATOR. A symbol that specifies a Boolean arithmetic operation.

LOGICAL PACKET. (MultiSite) The complete set of data required to create a new VOB replica, or to synchronize two or more existing replicas in a VOB family. A logical packet can encompass several physical files. See also *physical packet*.

LOST+FOUND. A subdirectory of a VOB's top-level directory, to which elements are moved if they are no longer cataloged in any version of any directory element. See *orphaned element*.

MAGIC FILE. A file used by the ClearCase *file typing* subsystem to determine the type of an existing file, or for the name of a new file. A magic file consists of an ordered set of file-typing rules.

MAIN BRANCH. The starting branch of an element's *version tree*. The default name for this branch is **main**.

MAKE MACRO. A parameter in a *makefile*, which can be assigned a string value within the makefile itself, in a *build options spec*, on the **clearmake** or **omake** command line, or by assuming the value of an environment variable. (**NOTE**: **omake** is supported only on Windows platforms.)

MAKEFILE. A text file, read by **clearmake** or **omake**, that associates *build scripts*, consisting of shell commands (executable commands), with *targets*. Typically, executing a build script produces one or more *derived objects*. (**NOTE**: **omake** is supported only on Windows platforms.)

Makefiles constructed for **clearmake** or **omake** need not include source-level dependencies (for example, header file dependencies), but they must include build-order dependencies (for example, that executable program **hello** is built using object module **hello.o**.

MAKEFILE DEPENDENCY. See *dependency.*

MASTER. (MultiSite) See *mastering replica*.

MASTERING REPLICA. (MultiSite) The mastering replica of a ClearCase object is the only replica at which the object can be modified or instances of the object can be created.

MASTERSHIP. (MultiSite) The ability to modify an object, or to create instances of a type object.

MERGE. The combining of the contents of two or more files or directories into a single new file or directory. Typically, when merging files, all the files involved are *versions* of a single file element. When merging directories, all *contributors* to the merge must be versions of the same directory element.

MERGE OUTPUT FILE OR DIRECTORY. The output of a *merge* operation. Once you have resolved any conflicting *differences* between *contributors*, you can save the merged contents into the merge output file or directory. Typically, the merge output file or directory is written to the CHECKEDOUT version of the contributor to which you are merging.

META-DATA. Data associated with an *object*, supplementing the object's *file system data*.

Example: The contents of a file version is a series of text characters. User-specified meta-data annotations attached to the file version includes *version labels*, *attributes*, and *hyperlinks*. ClearCase automatically maintains other meta-data for some objects, in the form of *event records* and *configuration records*.

METHOD. A program, subroutine, or utility that implements one of the functions of a *type manager*.

MINOR EVENT. An event whose *event record* is, by default, not displayed by the history browser or listed by the **lshistory** command.

MNODE. An MVFS-specific data structure, used to maintain information about an *MVFS object*. An mnode is attached to an operating-system *vnode*, as private data.

MODE. See *access mode*.

MOUNTED VOB. An *active* VOB.

MSDOS-ENABLED MODE. An operating mode for a VOB, wherein the VOB database tracks the number of lines in a text file, enabling correct file-size reporting on both UNIX and Windows NT systems.

MSDOS TEXT MODE. See *text mode*.

MULTIVERSION FILE SYSTEM (MVFS). A directory tree which, when activated (mounted as a file system of type MVFS) implements a ClearCase VOB. To standard operating system commands, a VOB appears to contain a directory hierarchy; ClearCase commands can also access the VOB's meta-data. Also, *MVFS file system* refers to a file system extension to the operating system, which provides access to VOB data. The MVFS file system is not supported on all ClearCase platforms.

MVFS CACHE. An in-memory data structure that speeds *MVFS* performance. Data on several kinds of file system resources are maintained, in separate caches.

MVFS OBJECT. A file or directory whose pathname is within a VOB (that is, whose pathname is within the directory tree beneath the top-level *VOB-tag*. A *non-MVFS object* has a pathname that is not within a VOB.

NAMESPACE. A file/directory name hierarchy. Different views can see different namespaces, because they can select different versions of directory elements. See *extended namespace*.

NETWORK PROVIDER. (Windows platforms only) A supplier — an extended file system or Windows NT Server, for example — of a complete networking system for Windows NT hosts. Because it functions as a network provider, the ClearCase MVFS deactivates VOBs and views at user login time. (**NOTE**: For UNIX platforms, see *virtual file system*.)

NETWORK REGION. A logical subset of a local area network, within which all hosts refer to VOB storage directories and view storage directories with the same network pathnames.

NOBODY. (UNIX platforms only) The username sometimes assigned to a remote process owned by the *root* user on the local host.

NON-CLEARCASE ACCESS. (UNIX platforms only) Access to ClearCase data from a host on which ClearCase has not been installed.

NONMASTERED CHECKOUT. An unreserved *checkout* performed on a branch that is not mastered by the local VOB replica.

NON-MVFS OBJECT. See *MVFS object*.

NONSHAREABLE DERIVED OBJECT. A derived object that cannot be *winked in* by other views. If your dynamic view is configured with the nonshareable DO property, the ClearCase build programs create DOs in the view, but do not write shopping information into the VOB.

NOTICE FORWARDER. (UNIX platforms only) One of the message-passing programs in an H-P SoftBench environment.

NULL-ENDED. A *hyperlink* that is connected to only one object, not two.

OBJECT. An item stored in a *versioned object base*— *elements*, *versions* of elements, *branches*, *derived objects*, *hyperlinks*, *locks*, *pools*, and *types*, and so on. An object can identified by an object-selector string, which includes a prefix indicating the kind of object, the object's name, and a suffix indicating the VOB in which the object resides. Examples: **lbtype:REL1@/vobs/vega** on UNIX and **lbtype:REL1@\vega** on Windows

OBJECT-ID (OID). A ClearCase-internal identifier for an object. See *UUID*.

OBJECT REGISTRY. A network-wide database, which records the actual storage locations of all VOB storage directories and all view storage directories. The **mktag**, **rmtag**, **mkview**, **rmview**,

mkvob, **rmvob**, **register**, and **unregister** commands add, delete, or modify registry file entries.

OBSOLETE OBJECT. An object that has been *locked* with the command **lock –obsolete**. By default, such objects are not listed by commands such as **lstype** and **lslock**.

OID. See *object-ID*.

OPLOG. (MultiSite) A list of all changes that have been made to a VOB's database.

OPLOG ENTRY. (MultiSite) The record of a single change to a VOB. Each oplog entry includes the identity of the originating replica and the *epoch number* of the operation.

ORDINARY TYPE OBJECT. A type object that is neither a *global type* object, nor a *local copy* of one. Prior to ClearCase V3, all type objects were ordinary.

ORIGINATING REPLICA. (MultiSite) The replica at which an operation was first performed.

ORPHANED ELEMENT. An element that is no longer cataloged in any version of any directory. Such elements are moved to the VOB's *lost+found* directory.

OWNER. The user who owns a VOB, a view, or an individual object. The user who creates an object becomes its initial owner.

OWNERSHIP-PRESERVING. (MultiSite) The subset of replicas within a VOB family whose elements share the same user and group identities. Only one such subset is allowed per VOB family.

PACKET. See *logical packet*, *physical packet*.

PAIRWISE DIFFERENCE. How the command-line output of cleartool commands (such as **diff, cleardiff**, **merge**, and **findmerge**) represent *differences* between *contributors* to the comparison or *merge*. For example, if three contributors (contrib2, contrib3, and contrib4) all differ from the base contributor (contrib1) in a particular section, the command-line output lists the contrib1-contrib2 difference, followed by the contrib1-contrib3 difference, followed by the contrib1-contrib4 difference.

PARALLEL BUILD. (UNIX platforms only) A *build* process in which multiple build scripts are executed concurrently. See also *distributed build*.

PARALLEL DEVELOPMENT. The concurrent creation of versions on two or more branches of an element.

PATHNAME. A sequence of directory names, perhaps ending with a simple filename. A pathname that begins with a backslash (\), or with a drive letter (**A:**, **B:**, **C:** ...) and backslash (on Windows) or a "/", indicating the root directory (on UNIX), is termed a *full pathname*. Any other pathname is termed a *relative pathname*. See *extended pathname*, *namespace*, *absolute VOB pathname*.

PATTERN. A character string that specifies one or more file and/or directory names. Examples:

UNIX:

```
/usr/project/.../include
*.c
lib/*.[ch]
```

Windows:

```
\proj_1\...\include
*.c
```

See *scope*, *version selector*, *config spec*.

PERL. A general-purpose scripting language (Practical Extraction and Report Language).

PERMISSION. Ability to perform an operation that modifies a file, directory, or other object.

PERSISTENT VOB / PERSISTENT VIEW. (Windows platforms only) A VOB or view that is automatically activated each time you login.

PHYSICAL PACKET. (MultiSite) A file containing one part of a *logical packet*.

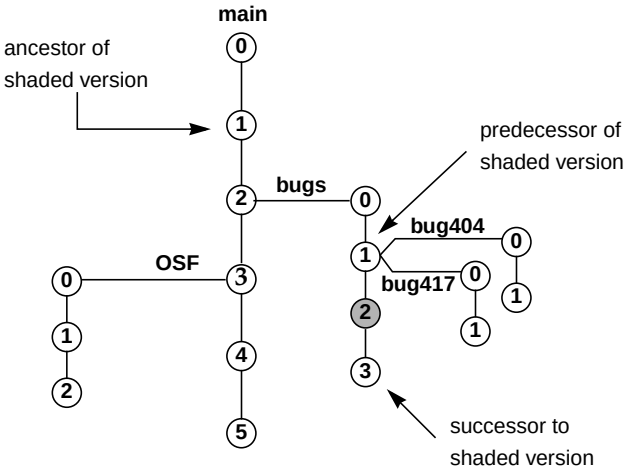
POOL. See *storage pool*.

POOL INHERITANCE. The feature by which a newly-created element is assigned to the same VOB storage pools as its parent directory element.

POST-OPERATION TRIGGER. A trigger that fires after the associated operation.

PRE-OPERATION TRIGGER. A trigger that fires before the associated operation, possibly canceling the operation itself.

PREDECESSOR VERSION / SUCCESSOR VERSION. A *version* of an *element* that immediately precedes another version in the element's *version tree*. If version X is the predecessor of version Y, then Y is the *successor* of X. If there is a chain of predecessors linking two versions, one is called an *ancestor* of the other.



PRIMARY GROUP. (Windows platforms only) The first group to which a user belongs. Typically, this domain-wide group assignment is made by the *administrator* using the User Manager for Domains utility. A user can belong to additional groups, as well.

PRIMARY REGISTRY SERVER HOST. See *registry server host*.

PRINCIPAL GROUP. (UNIX platforms only) The group to which a user belongs, by virtue of being listed in the password database. A user can belong to additional groups, as well.

PRIVATE STORAGE AREA. The directory tree (**.s**) in which view-private files, directories, and links are stored. By default, this is a subtree of the view storage directory, but ClearCase supports creation of remote private storage areas.

PRIVATE BRANCH. A *branch* on which a team member works in isolation from other members of the team. The branch is usually created at a stable *version* identified by a *label*. The team member makes private work available to others by merging versions on the private branch with versions on an *integration branch*.

PRIVATE VOB-TAG. See *public VOB-tag*.

PROCESS OF RESTORATION. (MultiSite) The special state into which the MultiSite **restorereplica** command places a VOB replica after it has been restored from backup. The replica remains in this state until its database has been made consistent (through synchronization updates) with the other replicas in its VOB family.

PROJECT. A ClearCase UCM object that contains the configuration information needed to manage a significant development effort, such as a product release. Use the project to set policies that govern how developers access and update the set of files and directories used in the development effort. A project includes one *integration stream*, which configures views that select the latest versions of the project's shared elements, and typically multiple *development streams*, which configure views that allow developers to work in isolation from the rest of the project team.

PROJECT CONFIGURATION FILE. (Attache) A *local file* containing a list of files to be used by the **File→Update Workspace** command.

PROJECT VOB (PVOB). A VOB that stores UCM objects, such as projects, streams, activities, and change sets. Every UCM *project* must have a PVOB. Multiple projects can share the same PVOB.

PROMOTION. Migration of a derived object from view storage (unshared or nonshareable) to VOB storage (shared).

PROMOTION LEVEL. A property of a UCM *baseline* that can be used to indicate the quality or degree of completeness of the *activities* and *versions* represented by that baseline. You can use promotion levels to define policy for a UCM *project*.

UCM provides an ordered set of default promotion levels, and also supports user-defined promotion levels.

The action of changing the promotion level of a baseline is called promoting or demoting the baseline.

PROPERTIES BROWSER. A graphical tool that enables you to examine the properties of a VOB database object, including its meta-data annotations.

PROTECTION MODE. See *access mode*.

PRUNE. (hyperlink browser) Removing all objects to the right of the selected object.

PSEUDOTARGET. A *makefile target* that is not a file system object.

PUBLIC VOB-TAG. All VOBs with public *VOB-tags* get mounted automatically by the **cleartool mount -all** command. (On UNIX, the standard ClearCase scripts run this command as part of the boot process that starts ClearCase and so all public VOBs are typically always mounted.) Creating a VOB with a public VOB-tag requires that you know the VOB-tag password. (See also *VOB-tag password file*.)

On UNIX platforms, any user can mount or unmount a public VOB by naming it explicitly using the **cleartool mount** or **unmount** command. If a VOB's VOB-tag is private, only the VOB's owner or the root user can mount or unmount the VOB.

On Windows platforms, any user can mount or unmount any VOB, public or private, by naming it explicitly using the *cleartool mount* or *unmount* command. The private designation means only that the VOB must be mounted explicitly by name.

PUT. (Attache) The command that *uploads* one or more files from your *workspace* to its associated *view*.

PVCS. The Polytron Version Control System.

PVOB. See *Project VOB (PVOB)*.

QUERY LANGUAGE. A collection of predicates and logical operators that allows selection of elements, branches, and versions based on their associated meta-data. This language can be used in *config specs*, in the ClearCase **find** command, and in the **–version** option to many commands.

RCS. The Revision Control System.

REBASE. A ClearCase UCM operation that makes your development work area current with the set of versions represented by a more recent *baseline* in the *integration stream*.

RECOMMENDED BASELINE. The set of baselines that the project team should use to *rebase* their *development streams*. In addition, when developers join a project, their development work areas are initialized with the recommended baselines.

The recommended baselines represent a system configuration, or set of components, that has achieved a specified promotion level. A baseline becomes part of the set of recommended baselines when the project manager promotes it to a certain promotion level, for example, TESTED.

RECORD. See *event record*, *configuration record*.

RECOVERY INCARNATION. (MultiSite) A numerical value associated with a replica; it is incremented if the replica undergoes restoration from backup. This value is attached to outgoing packets and is used by other replicas to determine whether the packet was sent before or after the restoration (and, therefore, if they should import it).

REFERENCE COUNT. The number of references to a *derived object* from multiple *views*. When *scrubbing*, the reference count is evaluated to determine whether to delete the derived object to free storage space.

REFERENCE TIME. See *build reference time*.

REFINEMENT. See *supertype*.

REGION. See *network region*.

REGISTER. Create an entry in the *tags registry* and/or the *object registry*, for a view or a VOB.

REGISTRY REGION. See *network region*.

REGISTRY SERVER HOST. The host on which all ClearCase data storage areas (all VOBs and views) in a local area network are centrally registered. ClearCase supports a primary and backup registry server host. The backup host is used if the primary host becomes unavailable. See the **rgy_backup** reference page.

RELATIVE PATHNAME. A pathname that does not begin with a "/" on UNIX or with a "\", or with a drive letter and backslash (**Z:**).

RELEASE AREA. A directory structure, accessible by all client hosts, containing the software distribution for one or more products in the ClearCase product family. In particular, a *merged release area* holds the distributions for two or more products.

RELEASE HOST. The host on which the ClearCase software is unloaded from the distribution medium.

RELOCATE SET. The set of file and directory elements that are actually moved by the **relocate** command; a subset of the *selection set*.

REMOTE COMMAND. (Attache) A command that affects only ClearCase storage (*view* or *VOB*), not your *workspace*.

REMOTE DELIVER. (A variation of the UCM *deliver* operation. UCM uses remote deliver in a ClearCase MultiSite configuration when the project's *integration stream* and individual *development stream* are mastered in different replicas. When these conditions exist, a developer starts the deliver operation at the remote site but leaves it in a posted state. The project manager must complete the deliver operation at the integration stream's replica site.

REMOTE STORAGE POOL. See *storage pool*.

REPLICA. (MultiSite) An instance of a VOB, located at a particular *site*. A replica consists of the VOB's database, along with all of the VOB's data containers.

REPLICA-CREATION PACKET. (MultiSite) A logical packet that contains the data for creating one or more new replicas within a VOB family.

REPLICA OBJECT. (MultiSite) The VOB database object that records the existence, name, site location, and other details of a particular VOB replica.

REPLICA UUID. (MultiSite) See *VOB family.*

REPLICATED VOB. (MultiSite) A VOB for which two or more replicas currently exist.

REROOTING. (hyperlink browser) Removing all objects to the left of the selected object.

RESERVED CHECKOUT. See *checkout*.

RESTORATION UPDATE PACKET. (MultiSite) A packet that contains specially requested data from a VOB replica, for use in restoring the database of another replica.

RESTRICTION LIST. A specification of which *type objects* are to be associated with a *trigger type*.

RESYNCHRONIZING THE WORKSPACE. (Attache) Using a *get* or **find –get** command to ensure that your *workspace* contains the same data as its associated *view*.

RETURN BAY. (MultiSite) A directory that holds incoming or outgoing MultiSite packets that are in the process of being returned to their origin because they could not be delivered to all specified destinations.

REUSE. During a build in a view, using an existing derived object in that view instead of winking in a DO from another view or rebuilding the DO.

ROOT. (UNIX platforms only) The privileged superuser on a UNIX host.

S-FILE. (UNIX platforms only) A file in which SCCS stores all versions of a versioned file.

SAME-VOB HYPERLINK. A *hyperlink* that connects two objects in the same VOB.

SCHEMA. The format of a database.

SCHEME FILE. (UNIX platforms only) A file that contains a collection of X Window System resources, which control various aspects of GUI applications.

SCOPE. The part of a *config spec* rule that restricts it to a particular kind of file system objects. See *config spec*, *pattern*, *version selector*.

SCRUBBING. Freeing storage space by removing objects that are no longer used:

- Discarding *data container* files from *cleartext pools* and *derived object storage pools*, performed by the **scrubber** utility.
- Discarding *event records* and MultiSite *oplog entries* from a *VOB database*, performed by the **vob_scrubber** utility.
- Removing *derived object* containers from the *view storage directory*, performed by the **view_scrubber** utility.

SELECTION EXPRESSION. An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one). The expression consists of *selection operators* and *logical operators*.

SELECTION OPERATOR. See *selection expression*.

SELECTION SET. The set of file and directory elements that are candidates for processing by the **relocate** subcommand, as specified by command arguments.

SELECTIVE MERGE. A *merge* that includes the changes contained in only a specified subset of *ancestors*. For example, if you are merging a *contributor* version /main/bugfix/4, but do not want the changes in the /main/bugfix/3 version of that contributor, a selective merge allows you to include the changes in versions 1, 2, and 4 on the /main/bugfix branch, but to omit the changes in the /main/bugfix/3 version.

SELF-MASTERING REPLICA. (MultiSite) A replica that masters its own *replica object*.

SEMI-LIVE BACKUP. See *VOB database snapshot*.

SERVER. See *albd_server*, *view_server*, *vob_server*, *workspace helper*.

SETGID, SETUID. The UNIX facility by which the operating system process in which a program runs gets its group-ID (setGID) or its user-ID (setUID) from the file in which the program is stored, rather than from user who invoked the program.

SET VIEW. (noun) (UNIX platforms only) The *view context* of a process, established by using the **setview** command. Setting a view creates a process in which all standard pathnames are resolved in the context of a particular view. See *working directory view.*

SHAREABLE DERIVED OBJECT. A derived object that can be *winked in* by other views.

SHARED DERIVED OBJECT. A derived object whose data container is located in a VOB's *derived object storage pool*. The DO may be referenced by multiple views.

SHARED TYPE OBJECT. (MultiSite) A *type object* whose instances can be managed at any replica. Creation of a new instance is controlled by the *mastership* of the object to which the new instance will be attached.

SHARENAME. (Windows platforms only) The name assigned to a drive or directory that is shared using the **Sharing** tab of the **Properties** dialog box in the Windows Explorer (or **net share** on the command line). A sharename is the second component in a *UNC name*.

SHIPPING ORDER. (MultiSite) A file that contains information for use by the MultiSite *store-and-forward* facility to deliver an update packet.

SIBLING. During a build, a derived object created by the same build script as another derived object. In MultiSite, a replica's siblings are the other replicas in its VOB family.

SID. (Windows platforms only) Security Identifier. A variable-length structure that uniquely identifies a user or group across all Window NT systems. The primary datum in a SID is a 48-bit value: the first 16 bits identify the SID-issuing agency — typically a Windows NT Server domain — and the remaining 32 bits identify a user or group within that issuing agency.

SINGLE-COMMAND MODE. The usage mode for a CLI tool in which you specify the tool's name (for example, **cleartool**), a subcommand, options, and arguments at an operating system command prompt. After executing that one (sub)command, the CLI tool exits.

SITE. (MultiSite) A location where one replica in a VOB family resides.

SNAPSHOT VIEW. A *view* that contains copies of ClearCase *elements* and other file system objects in a directory tree. You use an update tool to keep the view current with the VOB (as specified by the *config spec*).

SNAPSHOT VIEW UPDATE. A ClearCase operation you invoke to ensure that the versions in the view are the same versions in the VOB selected by the *config spec*. When necessary, an update operation copies files and directories from the VOB or removes or renames files and directories in the view.

SOFTBENCH. (UNIX platforms only) An interprocess messaging system.

SOURCE CONTROL. See *version control*.

SOURCE DEPENDENCY. See *dependency.*

SOURCE POOL. A *storage pool* for the *data containers* that store versions of file elements.

START-UP DIRECTORY. (Attache) The working directory of the Attache program when you start an Attache session. The default is *attache-home-dir***bin**. To change the startup directory in Windows 95: on the shortcut menu for the Attache icon, click **Properties**, then click the **Shortcut** tab and enter a directory in the **Start in** box. To change the startup directory in Windows 3.1: on the **File** menu for the Program Manager or File Manager, click **Properties**.

STORAGE BAY. (MultiSite) A directory that holds MultiSite *packets* and *shipping orders* for use by the *store-and-forward* facility.

STORAGE CLASS. (MultiSite) A user-specified name that is associated with certain information-delivery parameters by the MultiSite *store-and-forward* facility. For example, each storage class is associated with a particular *storage bay* (or set of bays) and a particular *expiration period*.

STORAGE POOL. A *source pool*, *derived object pool*, or *cleartext pool*. If it resides on a different host than the *VOB database*, it is termed a *remote storage pool*.

STORAGE REGISTRY. See *object registry.*

STORE-AND-FORWARD. (MultiSite) The MultiSite facility for transferring packets (or other files) among sites, either directly or through intermediate hops.

STRANDED CONFIGURATION RECORD. A *configuration record* (CR) that does not correspond to any existing *derived object*.

STRANDED DERIVED OBJECT. A *derived object* that cannot be accessed, because the VOB directory in which it was created is not currently accessible or has been deleted.

STRANDED VIEW-PRIVATE FILE. A *view-private file* that cannot be accessed, because it has no name in the VOB namespace, as currently constructed by your view. The **lsprivate** command lists such files; the **recoverview** command salvages them.

STREAM. A ClearCase UCM object that determines which versions of elements appear in any view configured by that stream. Streams maintain a list of baselines and activities. A project contains one *integration stream* and typically multiple *development streams*.

SUBBRANCH. A branch of an *element*'s *version tree* other than the main *branch.*

SUBSESSION. A *build session* that is started while a higher-level *build session* is active.

SUBTARGET. In a hierarchical build, a *makefile* target upon which a higher level target depends. Subtargets must be built, reused, or *winked in* before higher-level targets.

SUBTRACTIVE MERGE. A *merge* operation that removes the contributions of one or more *ancestors* of a *version*.

SUCCESSOR VERSION. See *predecessor version*.

SUPERTYPE. An *element type* that is used as the basis for defining another element type (said to be a *refinement* of the supertype).

SYNCHRONIZATION UPDATE PACKET. (MultiSite) See *update packet*.

TAGS REGISTRY. A network-wide database, which records the globally-valid access paths to all VOB storage directories (or all view storage directories), along with the *VOB-tags* (or *view-tags*) with which users access these data structures.

TARGET, TARGET REBUILD. A target rebuild is the execution of a *build script* associated with a particular *target* in a *makefile*. Each target rebuild produces derived objects along with a *configuration record*, which includes an *audit* of the files involved in the actual target rebuild.

TEXT_FILE. A ClearCase element type that uses *delta* management to combine all versions of an element into a single data container. The associated *type manager* is named *text_file_delta*.

TEXT_FILE_DELTA. The *type manager* for the predefined *text_file* element type.

TEXT MODE. A view is assigned a *text mode* when it is created. The text mode determines how the view handles line termination sequences — **unix**: <NL>, **msdos**: <CR><NL>. See **mkview**.

TEXT-ONLY. A hyperlink for which there is no *to* object, only a text annotation on the *from* object.

TIME RULE. A separate config spec rule that specifies a time to which the special version label **LATEST** should evaluate in all subsequent rules; or, a clause that sets the **LATEST** time within an individual rule.

TITLE BAR. The standard Microsoft Windows colored bar at the top of a window. For example, the Attache window's title bar shows the product name, along with the name of the *current workspace* and the host where the *workspace helper* program is running.

TO-OBJECT. See *hyperlink*.

TO-TEXT. A string-valued attribute attached to a hyperlink object, conceptually at its *to* end.

TRANSCRIPT PAD. (Attache) The *Command window* also acts as a historical record of command execution. In this window, the commands you enter are interspersed with the commands' output.

TRANSLATION FILE. A file that controls the mapping of symbols, branch names, and label names to ClearCase branch and label names during export of ClearCase, PVCS, RCS, SCCS, or SourceSafe data. Use this file to enforce naming consistency over multiple invocations of the exporter program. See the **clearexport_ccase**, **clearexport_pvcs**, **clearexport_rcs**, **clearexport_sccs**, and **clearexport_ssafe** reference pages. (**NOTE**: **clearexport_ssafe** is supported only on Windows platforms.)

TRANSPARENCY, TRANSPARENT ACCESS. The ClearCase feature that enables standard programs to access versioned files and directories using standard pathnames.

TRIGGER. A monitor that specifies one or more standard programs or built-in actions to be executed automatically whenever a certain ClearCase operation is performed. See *pre-operation trigger*, *post-operation trigger*, *trigger type*.

TRIGGER INHERITANCE. The process by which *triggers* in the *inheritance list* of a *directory element* are automatically attached to new elements created within the directory.

TRIGGER TYPE. An object through which triggers are defined. Instances of an "element" *trigger type* can be attached to one or more individual elements ("attached trigger"). An "all-element" trigger type is implicitly attached to all elements in a VOB. A "type" trigger type is attached to a specified collection of *type* objects.

TRIVIAL MERGE. A *merge* in which the *base contributor* and the *contributor* to which you are merging are the same. In this case, all *differences* are between the base contributor and the contributor

from which you are merging. ClearCase just copies the contributor from which you are merging to the contributor to which you are merging.

TYPE. A *type* object defines a ClearCase data structure. Users can create instances of these structures: meta-data annotations are placed on objects by creating instances of label types, attribute types, and hyperlink types. Each version-controlled file and directory is an instance of an element type; each branch is an instance of a branch type; and so on.

TYPE OBJECT. An object that defines the characteristics of (acts as the prototype for) an entire category, or class, of data items. ClearCase defines label types, branch types, hyperlink types, and so on.

TYPE MANAGER. A set of routines (*methods*) that stores and retrieves versions of file elements from disk storage. Some type managers include methods for other operations, such as comparison, merging, and annotation. See the **type_manager** reference page.

TYPE TRIGGER TYPE. A *trigger type* that is associated with (and thus, monitors changes to and usage of) one or more *type objects*.

UMASK. The UNIX file creation mode mask, a three-digit octal number. The three digits (each zero by default) refer to read/write/execute permissions for owner, group, and others, respectively. Each digit is subtracted from the corresponding digit specified by the system for the creation of a file. (See **creat(2)** on UNIX platforms.) For example, umask `002` removes write permission for others— files normally created with mode `777` become mode `775`; files created with mode `666` become mode `664`. Note that the default umask is `022`, but ClearCase strongly recommends a umask of `002` (or, simply, `2`); otherwise, VOB, view, and derived object sharing can be problematical.

If you are operating in a mixed UNIX/Windows NT environment, accessing UNIX VOBs and views via a PC-based NFS client product, configure the PC-side NFS umask to `002`.

UNC NAME. (Windows platforms only) A convention for naming shared resources. The UNC name for a shared resource (file, directory, printer, etc.) has the following form:

*hostname**sharename**rest-of-path*

Use UNC names to specify VOB and view storage directories in **mkvob**, **mkview**, and **mktag** commands. See also *sharename*.

UNCHECKOUT. The act of cancelling a *checkout* operation.

UNIDIRECTIONAL. See *hyperlink*.

UNIFIED CHANGE MANAGEMENT (UCM). An out-of-the-box process, layered on base ClearCase and ClearQuest functionality, for organizing software development teams and their work products. Members of a project team use *activities* and *components* to organize their work.

UNIVERSAL UNIQUE IDENTIFIER. See *UUID*.

UNIX TEXT MODE. See *text mode*.

UNLOAD. Remove information about an element from a snapshot view, and delete the version from the view. To unload a directory element, ClearCase

- Recursively deletes all *loaded* elements.
- Renames the directory to *directory-name*.**unloaded**, thus preserving all *view-private files* and *view-private directories*.

UNREGISTER. See *storage registry.*

UNRESERVED CHECKOUT. See *checkout*.

UNSHARED DERIVED OBJECT. A derived object that has never been *winked in* to another view.

UNSHARED TYPE OBJECT. (MultiSite) A *type object* whose instances can be created and managed only at its mastering replica.

UPDATE PACKET. (MultiSite) A logical packet that contains data for synchronizing some or all of the existing replicas in a VOB family.

UPDATE A SNAPSHOT VIEW. See *snapshot view update.*

UPLOAD. (Attache) Transfer data from Attache to the *workspace helper host*.

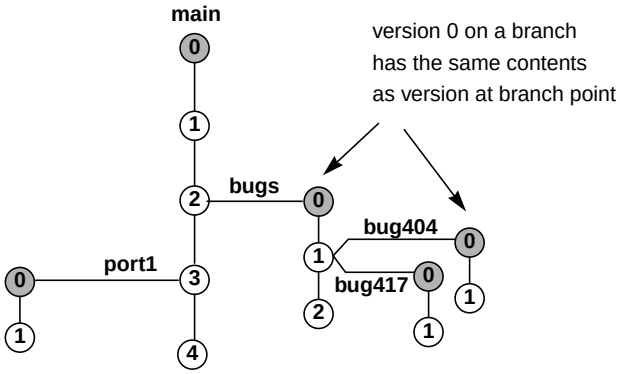
UPLOADING FILES. (Attache) Copying files from your *workspace* to its associated *view.*

USER PROFILE. A file that stores specifications for comment handling by individual **cleartool** subcommands.

UUID. Universal unique identifier. ClearCase uses UUIDs to track VOBs, views, and the objects they contain.

VERSION. An *object* that implements a particular revision of an *element*. The versions of an element are organized into a *version tree* structure. Also: checked-out version can refer to the *view-private file* that corresponds to the object created in a VOB database by the **checkout** command.

VERSION 0. The original *version* on a *branch*. It is automatically created when the branch is created, and has the same contents as the version at the branch point. Version 0 on the main branch is defined to be empty.



VERSION CONTROL. The discipline of tracking the version evolution of a file or directory.

VERSION-EXTENDED NAMESPACE. See *extended namespace*, *VOB-extended namespace*.

VERSION-EXTENDED PATHNAME. A pathname that explicitly specifies a version of an element (or versions of several elements), rather than allowing version-selection to be performed automatically by a view.

VERSION-ID. A *branch pathname* and *version number*, indicating a version's exact location in its version tree. Examples:

UNIX:

```
/main/4
/main/rel2_bugfix/2
/main/bugs/bug405/9
```

Windows:

```
\main\4
\main\rel2_bugfix\2
\main\bugs\bug405\9
```

VERSION LABEL. An instance of a label type object, supplying a user-defined name for a version. See *object*, *meta-data*.

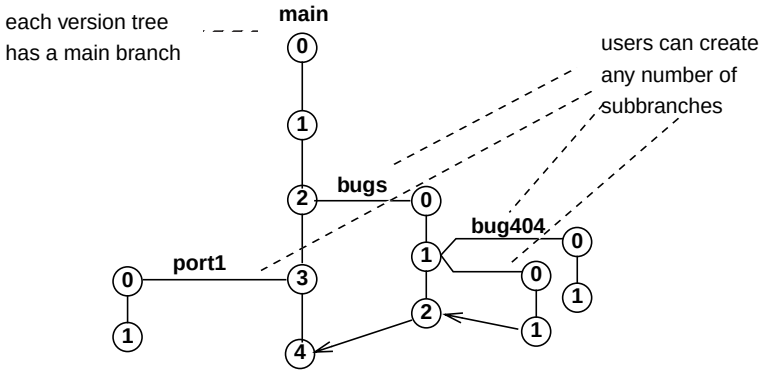
VERSION-NUMBER. The ClearCase-assigned integer that identifies the position of a version on its branch.

VERSION SELECTION. The process of choosing a specific version from an element's version tree. A *view* has several mechanisms that perform version selection. Users can select versions with *version-extended pathnames* and with the ClearCase *query language*.

VERSION-SELECTION RULE. A statement in the *config spec* that specifies a version of an element to be selected by the view. See also *load rule*.

VERSION SELECTOR. A specification (for example, **/main/17** (on UNIX platforms), **\main\17** on Windows platforms), or **RLS4.3-BETA** (on all platforms)) that identifies particular versions of one or more elements. See *version selection*, *scope*, *pattern* and *configuration specification*, and the **version_selector** reference page.

VERSION TREE. The hierarchical structure in which all the *versions* of an *element* are (logically) organized. When displaying a version tree, ClearCase also shows *merge* operations (indicated by arrows in the illustration).



VERSIONED OBJECT BASE (VOB). A repository that stores *versions* of *file elements*, *directory elements*, *derived objects*, and *meta-data* associated with these objects. With MultiSite, a VOB can have multiple *replicas*, at different *sites*.

VIEW. A ClearCase object that provides a work area for one or more users — to edit source versions, compile them into object modules, format them into documents, and so on. Users in different views can work with the same files without interfering with each other. For each *element* in a *VOB*, a view's *config spec* selects one version from the element's *version tree*. Each view can also store *view-private files* and *view-private directories*, which do not appear in other views. There are two kinds of views: *snapshot views* and *dynamic views*.

VIEW CONTENTS. (Attache) One of the two top-level entries listed in the *File Browser* for the current workspace. When expanded it shows all the *VOBs* currently active on the *workspace helper host*, as seen through the *view* associated with the *current workspace*.

VIEW CONTEXT. The view (if any) that will be used to resolve a pathname to a particular *version* of an *element*.

VIEW DATABASE. The database that tracks objects in a view.

VIEW-EXTENDED NAMESPACE. See *extended namespace*.

VIEW-EXTENDED PATHNAME. A pathname that begins with a *view prefix* (for example, **/view/alpha** on UNIX, or **M:\alpha** on Windows), specifying a particular view to be used for resolving element names to particular versions.

VIEW HOST. A host on which one or more *view storage directories* reside.

VIEW LOG. A log file, located on a particular machine, that records errors in accessing the view storage areas on that machine.

VIEW OBJECT. An object stored in a *view*: a *checked-out version* of a file, an *unshared derived object*, a *nonshareable derived object*, or a *view-private file*, directory, or link. No historical information is retained for view objects. See *VOB object*.

VIEW PREFIX. One or more components at the beginning of a pathname that specify a particular dynamic view. For example, on Windows, **M:\gamma** (or when **M:** is the current drive, simply **\gamma**), and on UNIX, **/view/gamma**. See also *viewroot directory*.

VIEW-PRIVATE DIRECTORY. A directory that exists only in a particular *view*, having been created with the standard **mkdir** command. A view-private directory is not *version-controlled*, except insofar as it is separate from private directories in other views.

VIEW-PRIVATE FILE. A file that exists only in a particular *view*. A private file is not *version-controlled*, except insofar as it is separate from private files in other views.

VIEW-PRIVATE OBJECT. A file or directory that exists only in a particular view. View-private objects are not *version-controlled*. See *view-private directory* and *view-private file*.

VIEW PROFILE. (Windows platforms only) A description of a ClearCase configuration that is shared by members of a team working on a project. A view profile includes the following information: the *VOBs* used by the project; the name of the *administrative VOB*, if any; a *config spec*; a list of *labels* identifying stable versions from which to create *private branches*; and the *branch type* of the *integration branch*. To use this shared configuration, a team member works in a *view* associated with the view profile.

VIEW REGISTRY. See *view storage registry, object registry, tags registry*.

VIEWROOT DIRECTORY. (The portion of an absolute path to an *element* that precedes the *view-tag* of a *snapshot view*. See also *dynamic-views root directory* and *view prefix*.

VIEW_SERVER. The daemon process that interprets a view's *config spec*, mapping element names into versions, and performs workspace management for the *view*.

VIEW STORAGE DIRECTORY. The directory tree ClearCase uses to maintain internal information about a view. Along with other files and directories, the view storage directory contains the *config spec* and the *view database*.

VIEW STORAGE REGISTRY. A file on the network's *registry server host* that records the *view storage directory* of every *view* in the network.

VIEW-TAG. The name with which users reference a *view*.

VIRTUAL FILE SYSTEM. (UNIX platforms only) An extension to the UNIX kernel, allowing alternative file systems to be implemented without revision to the kernel itself. (**NOTE**: For Windows platforms, see *network provider*.)

VNODE. An operating system kernel data structure, representing a file or directory. See also: *mnode*.

VOB. See *versioned object base*.

VOB BROWSER. A graphical application that administrators use to create, maintain, and control access to the VOBs in a local area network.

VOB DATABASE. The part of a *VOB storage directory* in which ClearCase *meta-data* and VOB objects (elements, branches, versions, and so on) are stored. This area is managed automatically by ClearCase's embedded database management software. The actual file system data, by contrast, is stored in the VOB's *storage pools*.

VOB DATABASE SNAPSHOT. A copy of a *VOB database*, made by the **vob_snapshot** utility, which enables a *VOB storage directory* to be backed up without locking the VOB.

VOB-EXTENDED NAMESPACE. An extension to the operating system's file naming scheme, which allows any historical *version* of an *element* to be accessed directly by any program. The extension also provides access to the *meta-data* (but not the file system data) of all of a VOB's existing *derived objects*.

VOB FAMILY. (MultiSite) The set of all *replicas* of a particular VOB. All the replicas share the same *VOB family UUID*; each replica has its own *VOB replica UUID*.

VOB HARD LINK. A name, cataloged in a (version of a) directory element, for an element. Typically, the first such link is called the element's name; the term *VOB hard link* is used to refer to any additional names for the element.

VOB HOST. A host on which one or more *VOB storage directories* reside.

VOB LINK. A *VOB symbolic link* or *VOB hard link*.

VOB MOUNT POINT. The directory on which a *VOB storage directory* is mounted. All UNIX commands, and most ClearCase commands, access a VOB through its mount point. (**NOTE**: For Windows platforms, see *VOB-tag*.)

VOB OBJECT. An object stored in a *VOB*: *element*, *version* of element, *type*, *hyperlink*, *derived object*, and so on. See *view object*. Also, the object in a VOB database that records the existence and identity of the VOB itself.

VOB OWNER. Initially, the user who created a VOB with the **mkvob** command. The ownership of a VOB can be changed subsequently, with the **protectvob** command. Replicas at different sites may or may not have the same owner.

VOB REGISTRY. See *VOB storage registry, object registry, tags registry.*

VOB REPLICA. (MultiSite) See *replica*.

VOB REPLICA OBJECT. (MultiSite) See *replica object*.

VOB ROOT DIRECTORY. The top-level directory of a *VOB*, accessed through the pathname of its *mount point* (for example, **/vobs/project_x**) on UNIX platforms or through the pathname of its *VOB-tag* (for example, **\proj_vob**) on Windows platforms.

VOB_SERVER. The process that provides access to the data containers that store versions' file system data.

VOB STORAGE DIRECTORY. The directory tree in which a VOB's data is stored: elements, versions, derived objects, CRs, event history, hyperlinks, attributes, and other meta-data.

VOB STORAGE REGISTRY. A file on the network's *registry server host* that records the actual storage locations of all the VOBs in the network.

VOB SYMBOLIC LINK. An *object*, cataloged in a (version of a) directory element, whose contents is a pathname. ClearCase does not maintain a version history for a VOB symbolic link.

VOB-TAG. For UNIX platforms, this is the full pathname at which users access a *VOB*. The VOB storage directory is activated by mounting it as a file system of type MVFS at the location specified by its VOB-tag.

For Windows platforms, this is the *VOB*'s registered name and also its root directory — the pathname at which users access VOB data. A VOB-tag has a single component and begins with the backslash character (\). For example, **\myvob** and **\vob_project2** are valid VOB-tags.

VOB-TAG PASSWORD. The password required to create a *public VOB-tag*. The password is maintained on the ClearCase *registry server host*. (On Windows, the password is maintained in the Windows NT Registry on the registry server host.)

VOB-TAG PASSWORD FILE. A file used to validate the password entered by a user when creating a public VOB-tag.

VPATH. A *make macro* that specifies directories that will be searched for data during a build.

VTREE BROWSER. A graphical application with which you can examine the structure of an element's version tree.

WILDCARD. See *pattern*.

WINDOWS NT SERVER DOMAIN. (Windows platforms only) A group of Windows NT Server hosts that share a common security policy and user account database. A single Windows NT Server host acts as the primary domain controller. Each non-Server host can belong to at most one domain.

WINKIN. Causing a shareable derived object to appear in a view, even though its *file system data* is actually located in a VOB's *derived object storage pool*. Also, converting a nonshareable derived object to a shared derived object.

WORKING DIRECTORY. See *current working directory*.

WORKING DIRECTORY VIEW. The *view context* of a process, established by using the **cd** command to change the current working directory to a *view-extended pathname* or a *snapshot view*. On UNIX platforms, see *set view*.

WORKING DISCONNECTED. (Attache) Using data that has been *downloaded* to your PC using Attache, but without having an active Attache session.

WORKSPACE. (Attache) A private directory tree on your PC, which contains local copies of data files under ClearCase version control. The *current workspace* is the one active in the *Attache window*. The *workspace name* appears in the *title bar* and in the *command prompt*.

WORKSPACE CONTENTS. (Attache) One of the two top-level entries listed in the *File Browser* for the current Attache workspace. When expanded it shows the contents of the *current workspace*.

WORKSPACE HELPER. (Attache) A server program, running on some ClearCase host, that maintains a TCP/IP connection to the *Attache client* program running on your PC.

WORKSPACE HELPER HOST. (Attache) The ClearCase host on which the *workspace helper* program is running.

WORKSPACE NAME. (Attache) The name of a *workspace*, as specified in the **mkws** command (command-line interface) or the **File→New Workspace** command (graphical interface). Same as the *view-tag* of its associated view.

WORKSPACE REGISTRY. (Attache) A list, maintained by Attache, of all the *workspaces* that currently exist on your PC.

WORKSPACE ROOT. The top-level directory of a *workspace*. All *VOB-tags* appear as subdirectories of this top-level directory.

WORKSPACE STORAGE DIRECTORY. (Attache) The location on your PC where a *workspace* resides.

Z_TEXT_FILE_DELTA. The *type manager* for the predefined *compressed_text_file* element type. See *delta*.