Rational Software Corporation®

RATIONAL® CLEARCASE®

INTRODUCTION

UNIX/WINDOWS EDITION

VERSION: 2002.05.00 AND LATER

PART NUMBER: 800-025069-000

Rational®
the software development company

Introduction
Document Number 800-025069-000 October 2001
Rational Software Corporation 20 Maguire Road Lexington, Massachusetts 02421

Technical Acknowledgments

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

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

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

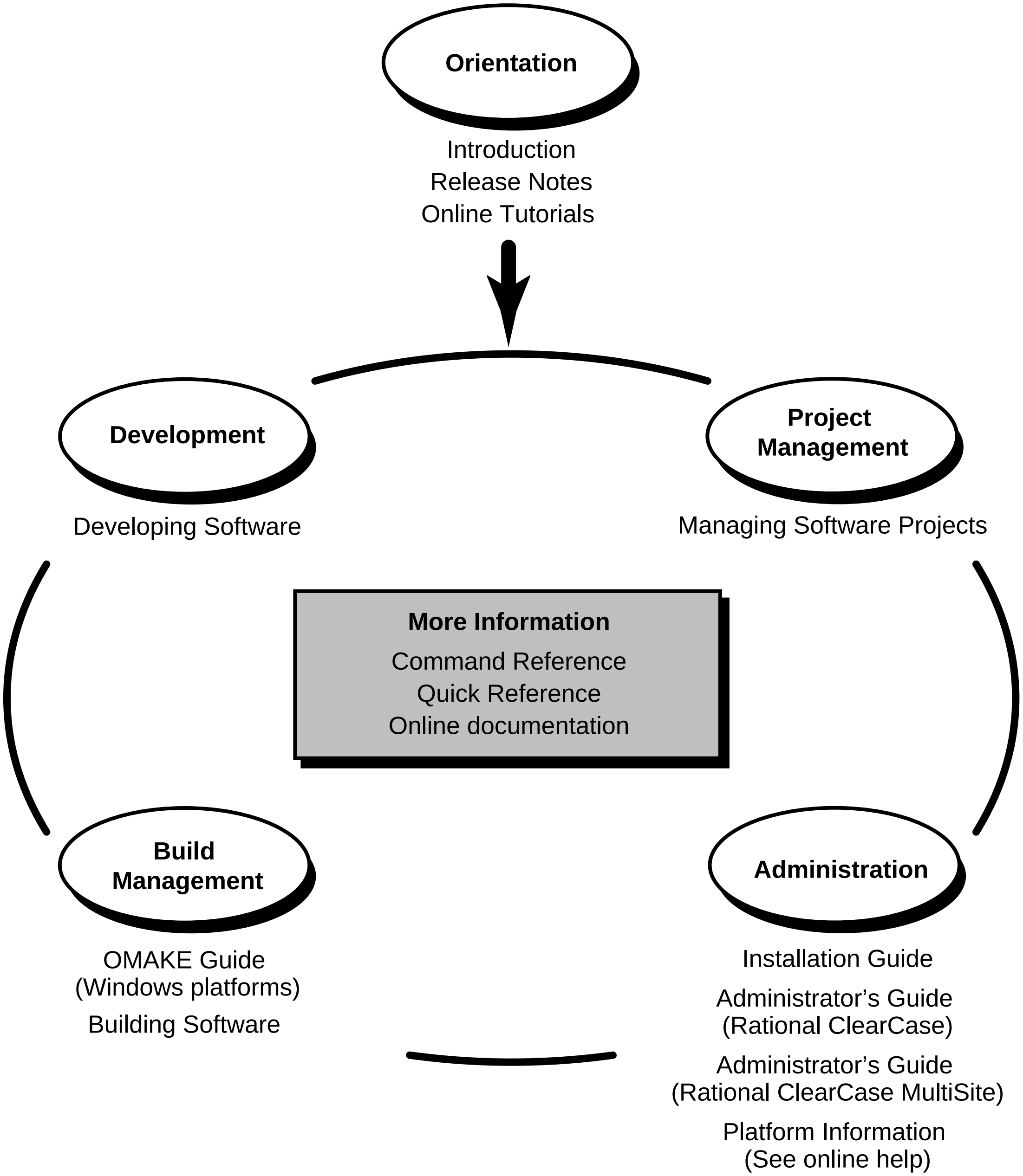
- **Integrator** (also called build engineer or release engineer) — builds and integrates the products of a software development project
- **Administrator** — configures and maintains the ClearCase infrastructure, including ClearCase VOBs, views, servers, and clients, for part or all of your organization

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

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

  lsc·heckout

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

Online Documentation

The ClearCase 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 uncheckout –help

```
Usage: uncheckout | unco [-keep | -rm] [-cact | -cwork ] pname ...
```

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** in the appropriate ClearCase folder off the **Start** menu.
- ➤ 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 Rational ClearCase, Rational ClearCase MultiSite, and Rational 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.

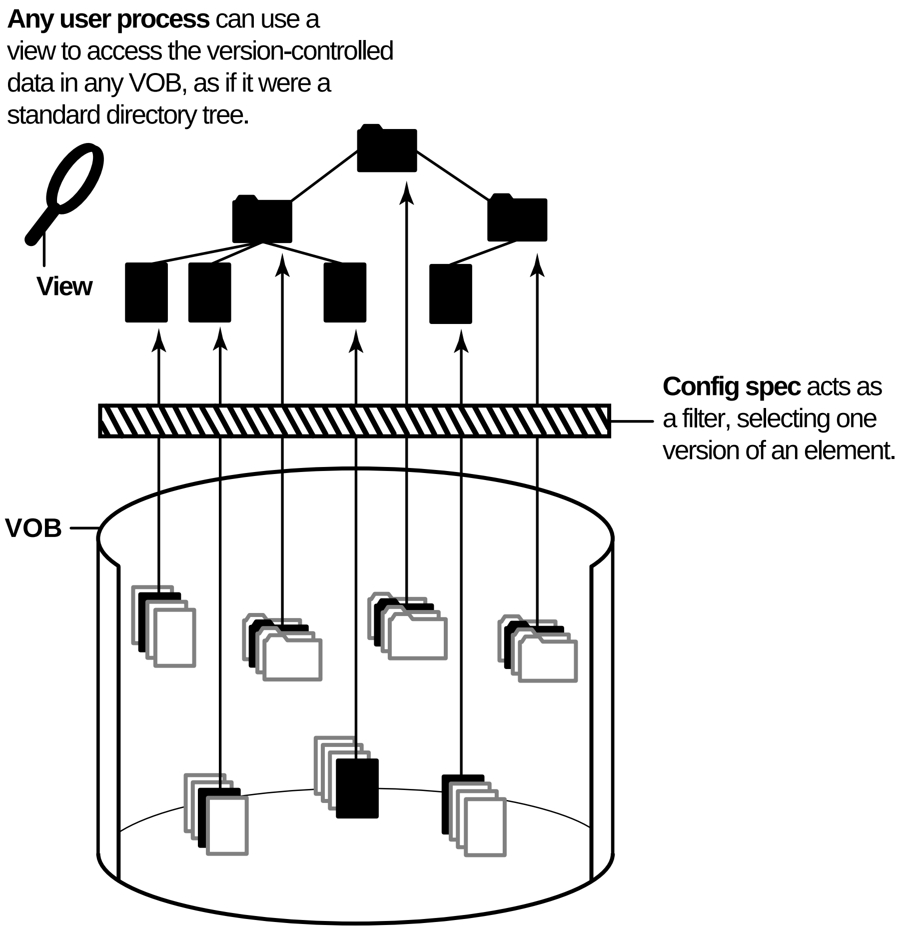
ClearCase is specifically designed to support parallel development, whether you are simply isolating the work of one developer from others on a small team, developing multiple releases in parallel using different teams, or sharing a source code base between multiple teams at geographically distributed 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 (Figure 1).

Figure 1 Accessing a VOB Using a View



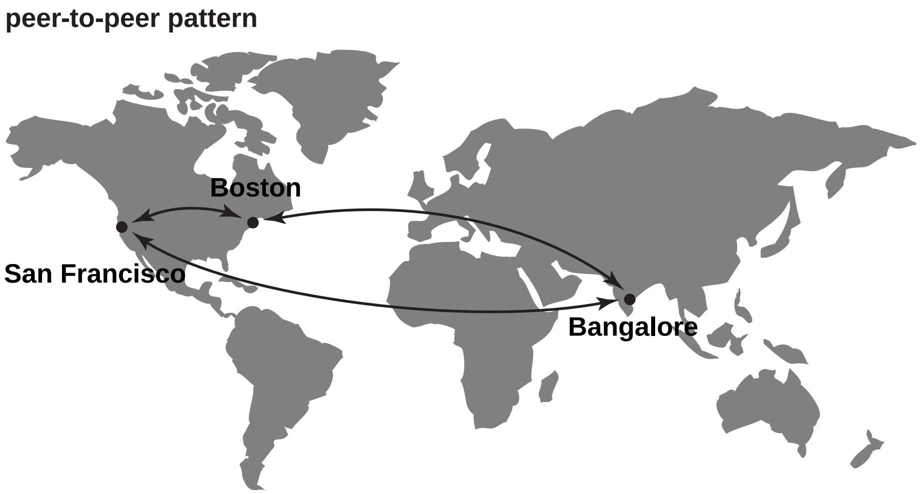
Like many configuration management systems, ClearCase uses a *checkout-edit-checkin model* to manage software changes. When you check out a file, ClearCase creates an editable copy, or *checked-out version*, in your view. When you check in a file, ClearCase creates a new, permanent version of the file 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 the *Administrator's Guide* for Rational ClearCase MultiSite 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

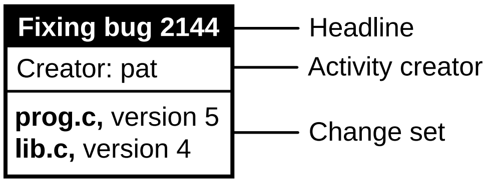
Rational Unified Change Management (UCM) combines ClearCase and ClearQuest to provide a complete, out-of-the-box, activity-based change management process.

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. A *project* is a ClearCase object that contains the configuration information (for example, components, activities, policies) needed to manage and track a significant development effort, such as a product release. Project managers 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.

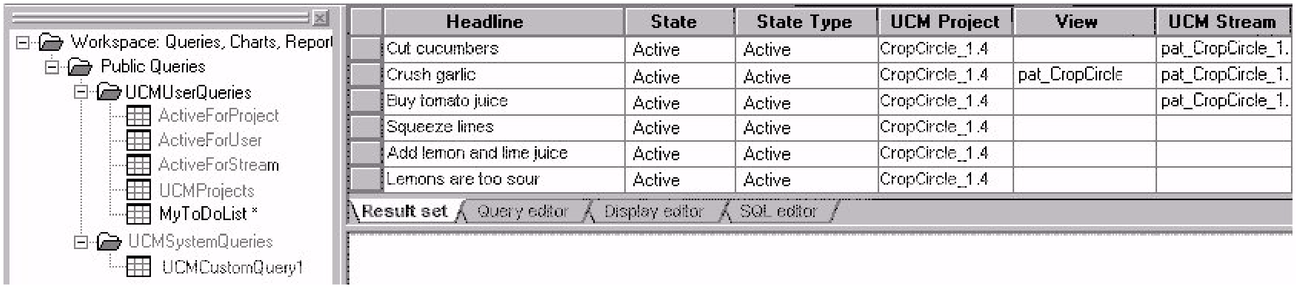
An *activity* is a ClearCase object that records the set of files (*change set*) that a developer creates or modifies to complete and deliver a development task, such as a bug fix (Figure 3).

Figure 3 An Activity



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 (Figure 4). Project managers can use ClearQuest queries, reports, and charts to monitor the progress of software development projects.

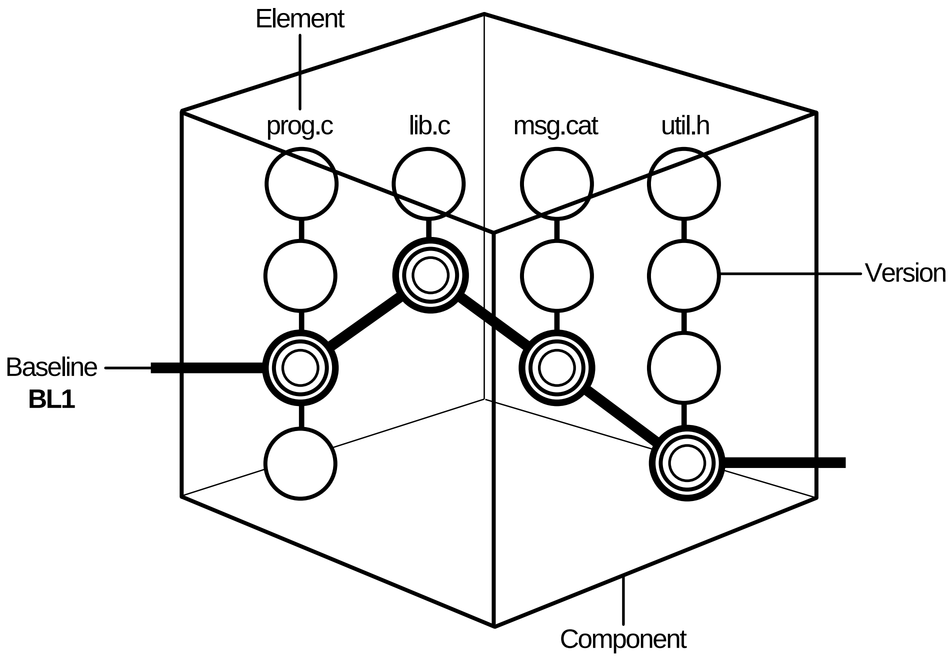
Figure 4 Using a ClearQuest To-Do List to Find UCM Activities



In UCM, a *component* is a group of related ClearCase directory and file elements that you develop, integrate, and release as a unit. Components constitute parts of a project, and projects often share components.

A *baseline* identifies one version of each element in a component that represents the integrated or merged work of team members (Figure 5). It represents a version of a component at a particular stage in project development, such as the first draft of a book, a beta release, or a final product release. Throughout the project cycle, the project manager creates baselines and changes their promotion level attributes to reflect project milestones.

Figure 5 Elements, Components, and Baselines



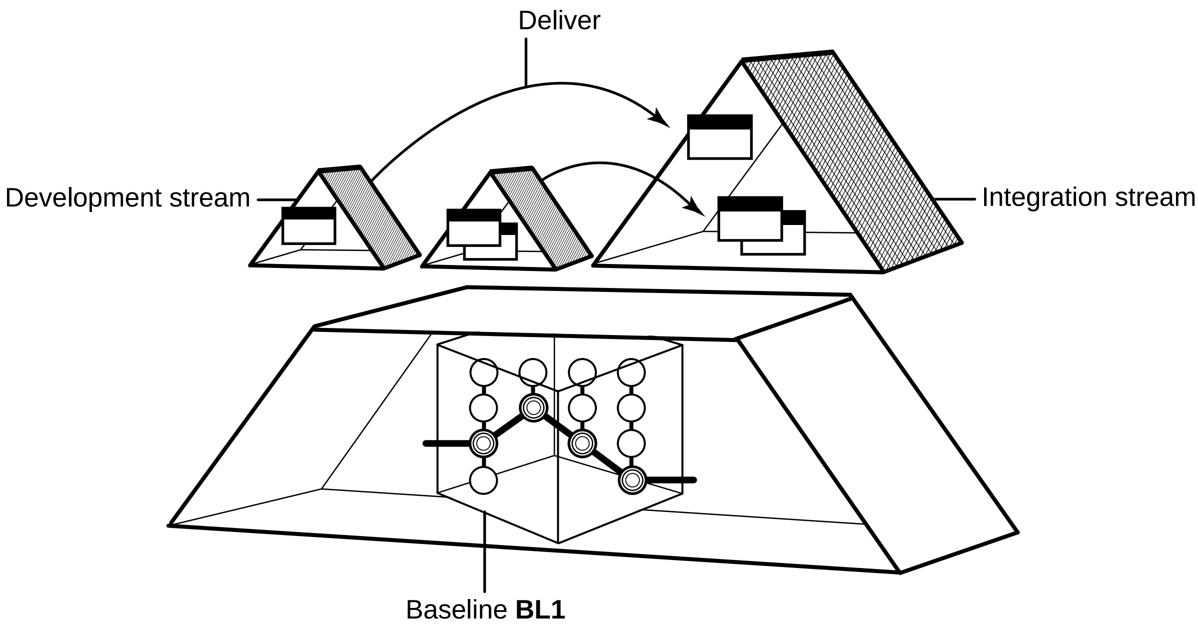
A UCM project includes a single *integration stream*, which configures ClearCase views that select the latest versions of a project's shared ClearCase elements. As a developer, when you join a project, ClearCase creates a *development stream* for you in the project. Development streams configure the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

When joining a project, you can populate your work area with the versions of directory and file elements represented by the project's recommended baseline. Alternatively, you can join the project at a feature-specific development stream level, in which case you populate your work area with the development stream's recommended baselines. This practice ensures that all members of the project team start with the same set of files.

If your project team works on multiple components, the project manager may want to use a composite baseline. A *composite baseline* is a baseline that selects baselines in other components. It provides a mechanism for holding together the recommended baselines of the project's components. By using a composite baseline in this manner, you can identify one baseline to represent the entire project.

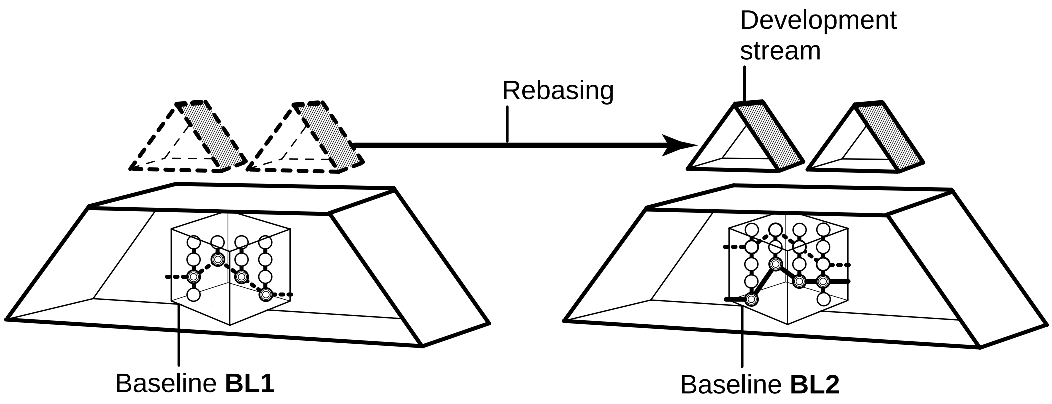
When you work on a particular activity, all changes to the files and directories in the components are associated with that activity in a change set. After you complete an activity, and build and test your work in your private work area, you share your work with the project team by *delivering* the activity from your development stream to the project's integration stream (Figure 6).

Figure 6 Delivering Activities from Development Streams to Integration Streams



Periodically, the project manager creates new baselines for the components used by the project. The new baselines incorporate work that developers have delivered since the last baselines were created.When the project manager recommends a particular baseline, you may choose to *rebase* your development stream to use the new baseline (Figure 7).

Figure 7 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 and ClearQuest. See *Developing Software*, *Managing Software Projects*, 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 Rational ClearCase, providing a high-level understanding of the installation process. It is not intended to be a complete installation guide. See the *Installation Guide* for the ClearCase Product Family 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, Rational ClearQuest, and Rational ClearCase 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* 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* 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* 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 the *Administrator's Guide* for Rational ClearCase MultiSite 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 the *Release Notes* for Rational ClearCase and ClearCase MultiSite 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 the *Installation Guide* for the ClearCase Product Family 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 the *Installation Guide* for the ClearCase Product Family 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 Rational 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* 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 of 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. A PVOB must exist before you can create a project.

Organizing Directories and Files into VOBs and Components

As the number of files and directories in your system grows, you need a way to reduce the complexity of managing them. Components are the UCM mechanism for simplifying the organization of your files and directories. The elements that you group into a component typically implement a reusable piece of your system architecture. By organizing related files and directories into components, you can view your system as a small number of identifiable components, rather than as one large set of directories and files.

The directory and file elements of a component reside physically in a VOB. The component object resides in a PVOB. Within a component, you organize directory and file elements into a directory tree. You can convert existing VOBs or directory trees within VOBs into components, or you can create a component from scratch. See *Managing Software Projects* for details.

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. For example:

- 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

Implementing Development Policies

UCM includes a set of policies that you can set on projects and streams to enforce development practices among members of the project team. By setting policies, you can improve communication among project team members and minimize the problems you may encounter when integrating their work. For example, you can set a policy that requires developers to update their work areas with the project's latest recommended baseline before they deliver work to the integration stream. This practice reduces the likelihood that developers will need to work through complex merges when they deliver their work. For a description of all policies that you can set in UCM, see *Managing Software Projects*.

In addition to the set of policies that UCM provides, you can create triggers on UCM operations to enforce customized development policies. A *trigger* is a monitor that specifies one or more standard programs or built-in actions to be executed whenever a certain ClearCase operation is performed.

See *Managing Software Projects* for details about creating triggers.

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, and 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 the

Administrator's Guide for Rational ClearCase for details about importing existing files and directories into ClearCase VOBs.

Applying a Label to the Initial Configuration

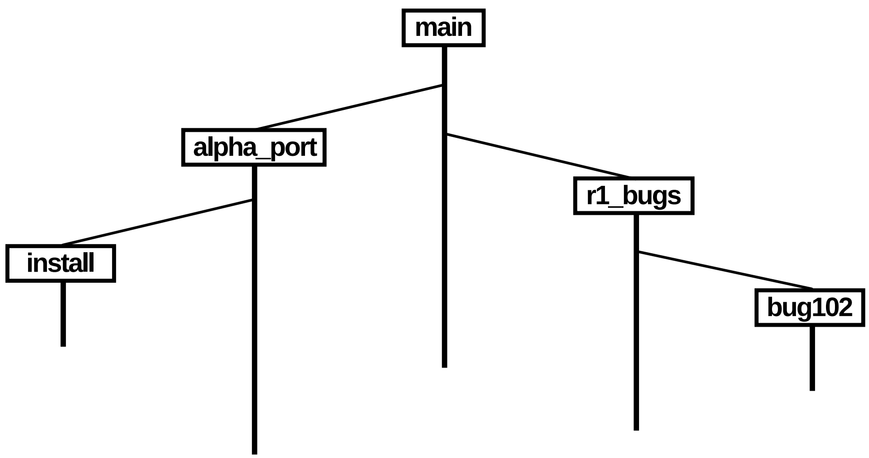
A *label* is a user-defined name that can be attached to a version.

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.

Establishing 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 8, 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 8 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 possible 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, the project manager can generate a variety of reports from this data and from the information that ClearCase captures in event records. ClearCase metadata that a project manager can use to define project policy includes the following:

- Version labels
- Attributes
- Hyperlinks
- Triggers
- Locks

See *Managing Software Projects* 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.

4 Developing and Building Software with ClearCase

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

See *Developing Software* for details about the development process and *Building Software* 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, you create your private work area and populate it with the contents of the project's baselines.

Shared and Private Work Areas

A work area consists of a view and a stream. A *view* is a directory tree that shows a single version of each file in your project. A *stream* is a ClearCase object that maintains a list of activities and baselines and determines which versions of elements appear in your view.

A project contains one *integration stream*, which records the project's baselines and enables access to shared versions of the project's elements. The integration stream and a corresponding integration view represent the project's shared work area.

Each developer on the project has a private work area, which consists of a development stream and a corresponding development view. The *development stream* maintains a list of the developer's activities and determines which versions of elements appear in the developer's view.

In the basic UCM process, the integration stream is the project's only shared work area. The project manager or lead developer may want to create additional shared work areas for developers who are working together on specific parts of the project. You can accomplish this by creating a hierarchy of development streams. For example, you can create a development stream and designate it as the shared work area for developers working on a particular feature. Developers then create their own development streams and views under the development stream for this feature. The developers deliver work to and rebase their streams to recommended baselines in the feature's development stream. See *Managing Software Projects* for details on development stream hierarchies.

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, such as fixing a bug.

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 either the project's integration stream or the feature-specific development 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 or feature-specific development stream. 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 or feature-specific development stream.

Delivering with MultiSite

If your project uses Rational ClearCase 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 either the project's integration stream or the feature-specific development 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 or feature-specific development 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 or feature-specific development stream to the development work area.

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, you create a ClearCase view and then change 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 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 the *Administrator's Guide* for Rational ClearCase MultiSite 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. The **make** program was developed for UNIX systems, and has been ported to other operating systems. To build software, you can use native **make** programs, third-party build utilities, your company's own build programs, or the ClearCase build tools **clearmake**, **omake** (Windows only), 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* 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 Rational ClearCase, using UCM and base ClearCase functionality.

See *Managing Software Projects* 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. As project manager, you are 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 you complete or cancel their remote deliver operations.

See the *Administrator's Guide* for Rational ClearCase MultiSite for details about MultiSite mastership. See *Managing Software Projects* for details about using MultiSite with UCM, including finding and completing posted deliveries.

Managing Baselines

This section summarizes how project managers create and recommend baselines.

Creating New Baselines

As developers deliver work to the integration stream, it is important that you, as the project manager, frequently make new baselines that incorporate the changes. If the project uses feature-specific development streams, you should perform this task on those streams as well as on the integration stream. In some environments, the lead developer working on a feature may assume the role of integrator for a feature-specific development stream. Developers can then rebase to the new baselines and stay current with changes in the project.

Recommending Baselines

As work on the project progresses and the quality and stability of the components improve, you can change a baseline's *promotion level* attribute to reflect important milestones. The promotion level attribute typically indicates a level of testing.

When a baseline passes the level of testing required to be considered stable, make it the recommended baseline. Developers then rebase their development streams to the recommended baseline. You can set a policy that requires developers to rebase their development streams to the recommended baseline before they deliver work. This policy helps to ensure that developers update their work areas whenever a baseline passes an acceptable level of testing.

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.

Using the ClearCase Report Builder and Report Viewer (Windows Only)

The ClearCase Report Builder and Report Viewer let you generate and view reports specific to your project environment. The Report Builder provides a set of reports organized by ClearCase object, such as project, stream, element, and view. In addition, you can customize the procedures used to generate reports.

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 *Establishing a Branching and Merging Strategy* on page 16, base ClearCase functionality uses branches to isolate parallel development efforts. At some point, as project manager, you integrate 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. After the work on the subbranch is deemed ready for integration, you merge the work from that branch to the main branch.

See *Managing Software Projects* 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 the *Administrator's Guide* for Rational ClearCase MultiSite for details about managing projects and integrating development work in ClearCase MultiSite.

Glossary

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

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

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 that contains global type objects, which are copied to client VOBs on an as-needed basis when users want to create instances of the type objects in the client VOBs. See also *auto-make-type*, *global type*, *local copy*.

ALBD_SERVER. Atria Location Broker Daemon. This 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.

ALL-ELEMENT TRIGGER TYPE. A trigger type that is 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.

ATTACHED LIST. The names of trigger types maintained for directory elements and file elements to control trigger firing. By default, a *trigger type* is added to the attached list when you attach a trigger to an element. If a trigger type is in the attached list of an element, operations on that element can cause that trigger to be fired. See also *inheritance list*.

ATTRIBUTE. A *metadata* annotation attached to an *object*, in the form of a name/value pair. Names of attributes are specified by *attribute types,* which users define; users can set the values of these attributes. 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).

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

AUTO-MAKE-TYPE. The ClearCase facility for 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 also *global type*, *local copy*.

BASE CONTRIBUTOR. In the comparison and merge tools, the *contributor* against which all other contributors are compared when reporting differences.

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.

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. An instance of a *branch type* for use in an element's version tree. By convention, all letters in the names of branch types are lowercase.

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

BRANCH TYPE. A *type object* that is used to identify a parallel line of development for an element. Each branch in an element's version tree is an instance of a branch type that exists in that element's VOB. A branch type can appear in a version selector to identify the branch on which a version resides.

BUILD. The process during which a ClearCase build program (**clearmake**, **clearaudit**, or **omake**) produces one or more *derived objects*. 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. On a client host, the MVFS performs an audit during execution of a ClearCase build program: **clearmake**, **omake**, **clearaudit**, or **abe**. 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 then 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 one 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 HOSTS FILE. (UNIX platforms only) A file that lists hosts to be used in a parallel build.

BUILD REFERENCE TIME. The time at which a *build session* 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.

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.The target must appear in a *makefile*, where it is associated with one or more *build scripts*.

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

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.

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* 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 check in a new version. An *unreserved* checkout does not. If several users have unreserved checkouts on the same branch in different views, the first user to check in creates the next version.

CLEARCASE ADMINISTRATORS 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 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*).

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 as a cache for the ASCII text extracted from *data containers*.

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.

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.

COMPOSITE BASELINE. A baseline that selects baselines from other components. You create a composite baseline by creating dependency relationships between the component that stores the composite baseline and the components whose baselines are selected by the composite baseline.

CONFIG SPEC. A set of configuration rules that specify 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 also *scope*, *version selector*, *version-selection rule*, and *load rule*.

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*.

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

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.

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.

CURRENT REPLICA. (MultiSite) The VOB replica in which you work. This is the replica in your ClearCase registry region.

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

DATA CONTAINER. **1)** 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*. **2)** A file in a *source pool* or *cleartext pool*, containing the data for one or more *versions* of a file *element*.

DELIVER. A ClearCase operation that allows developers to share their work with the project team by merging work from their own *development streams* to the project's *integration stream* or a feature-specific development 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*. Some type managers 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 produced by a **clearmake** or **omake** build or a **clearaudit** session. 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 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.

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

DIRECTORY ELEMENT. An *element* whose versions 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-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**.

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

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 (by default, drive M) 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* 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. A VOB object that is not visible because another object with the same name is currently selected by the view.

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

ELEMENT TYPE. A class of versioned file or directory objects. ClearCase supports predefined element types. Users can define additional types 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.

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, which indicate the current VOB replica's estimate of the 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 RECORD. An item in a VOB database that contains information about an operation that modified the VOB.

EXCEPTION LIST. The set of users to whom a *lock* or *trigger* does 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. Not writing information to the VOB speeds up the build, but it also means that the DOs cannot be winked in by other views.

EXTENDED NAMESPACE. The ClearCase extension of the standard Windows or UNIX pathname hierarchy. Each host has a view-extended namespace, which allows a pathname to access VOB data using any view that is active on that host. Each VOB has a *VOB-extended namespace*, which allows 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*.

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. A pathname that specifies a particular version is a *version-extended pathname*.

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

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

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

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.

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.

GLOBAL PATHNAME. A networkwide pathname for a *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 (client VOBs).

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. Metadata in a *VOB*, consisting of *event records* for 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 attributes assigned to the element and/or its versions, the hyperlinks attached to the element and/or its versions, and so on.

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. See also *global pathname*.

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.

HYPERLINK INHERITANCE. The 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*.

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, you can create incremental baselines faster than full baselines; however, ClearCase can look up the contents of a full baseline more quickly 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*.

INHERIT, INHERITANCE LIST. The names of trigger types maintained for a directory element to control *trigger inheritance*. By default, a trigger type is added to both the *attached list* and inheritance list of a directory element when you attach a trigger to that element. If a *trigger type* is in the inheritance list of a directory, newly created elements (but not existing elements) inherit that trigger.

INTEGRATION BRANCH. A *branch* that contains versions available to all members of a team. A team member often works 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, which 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.

INTEGRATION VIEW. A *view* attached to a UCM *project*'s *integration stream* or a feature-specific development stream. Use an integration view to build and test the latest work delivered to the stream. An integration view can be either a *dynamic view* or a *snapshot view.*

LABEL. An instance of a label type object, supplying a user-defined name for a version. See also *object*, *metadata*.

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

LOAD. To copy a version of an element to a snapshot view and keep track of the checkins, updates, and other ClearCase operations that affect 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 COPY. The copy of a *global type* object that is created in a client VOB during an *auto-make-type* operation.

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

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

LOGICAL PACKET. (MultiSite) The complete set of data required either 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.

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 *BOS file*, on the **clearmake** or **omake** command line, or by assuming the value of an environment variable.

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*.

MASTER REPLICA. (MultiSite) The master 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* must be versions of the same directory element.

MERGE OUTPUT FILE OR DIRECTORY. The output of a *merge* operation. After you resolve 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 checked-out version of the contributor to which you are merging.

METADATA. The data associated with an *object* that supplements its file system data. Some of this data is created by users; some of it is created during ClearCase operations on the object.

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 platforms.

MULTIVERSION FILE SYSTEM (MVFS). A directory tree that, when activated (mounted as a file system of type MVFS), implements a VOB. To standard operating system commands, a VOB appears to contain a directory hierarchy; ClearCase commands can also access the VOB's metadata. 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. A non-MVFS object has a pathname that is not within a VOB.

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.

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 your *current replica*.

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.

OBJECT. An item stored in a VOB. An object can be identified by an object-selector string, which includes a prefix that indicates the kind of object, the object's name, and a suffix that indicates the VOB in which the object resides. Examples: **lbtype:REL1@/vobs/vega** on UNIX and **lbtype:REL1@\vega** on Windows

OBJECT REGISTRY. A networkwide 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**.

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.

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.

PAIRWISE DIFFERENCE. How the command-line output of **cleartool** commands (such as **diff**, **cleardiff**, **merge**, and **findmerge**) represents differences between *contributors* to the comparison or *merge*.

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.

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

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

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 a chain of predecessors link two versions, one is called an *ancestor* of the other.

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

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.

PROJECT. A ClearCase UCM object that contains the configuration information needed to manage a significant development effort, such as a product release. 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 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. The 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.

PUBLIC VOB-TAG. All VOBs with public *VOB-tags* are 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 **umount** 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.

REBASE. A ClearCase operation that makes your development work area current with the set of versions represented by a more recent *baseline* in another stream, which is usually the project's *integration stream* or a feature-specific development stream.

RECOMMENDED BASELINE. The set of baselines that the project team uses 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.

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

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

REGISTRY SERVER HOST. The host on which all ClearCase data storage areas (all VOBs and views) in a local area network are registered. 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.

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.

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

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

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

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

SCRUBBING. The removal of objects that are no longer used to free storage space:

- The **scrubber** utility discards *data container* files from *cleartext pools* and *derived object storage pools*.
- The **vob_scrubber** utility discards *event records* and MultiSite *oplog entries* from a *VOB database*.
- The **view_scrubber** utility removes *derived object* containers from the *view storage directory*.

SELECTIVE MERGE. A *merge* that includes the changes contained in only a specified subset of *ancestors*.

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

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 also *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.

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

SIBLING. **1)** During a build, a derived object created by the same build script as another derived object. **2)** In MultiSite, the replicas in a *VOB family*.

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 that 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.

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

STORAGE BAY. (MultiSite) A directory that holds 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 *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* (also referred to as derived object storage pool), or *cleartext pool*. If it resides on a different host than the *VOB database*, it is a remote storage pool.

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

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*.

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.

TAGS REGISTRY. A networkwide 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 them.

TARGET REBUILD. 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.

TEXT MODE. A view is assigned a text mode when it is created. The text mode determines how the view handles line termination sequences

TIME RULE. **1)** A separate config spec rule that specifies a time to which the special version label **LATEST** should evaluate in all subsequent rules; **2)** A clause that sets the **LATEST** time within an individual rule.

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.

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

TRIGGER INHERITANCE. The process by which *triggers* in the *inheritance list* of a *directory element* are 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.

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

TYPE MANAGER. A set of routines 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.

TYPE OBJECT. An object that defines the characteristics of an entire category, or class, of data items.

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

UNC NAME. (Windows platforms only) A convention for naming shared resources. The UNC name for a shared resource (file, directory, printer, and so on) 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.

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.

UNLOAD. To remove information about an element from a snapshot view and delete the version from the view.

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 master replica.

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

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

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

VERSION ID. A *branch pathname* and *version number,* which indicate a version's exact location in its version tree.

VERSION LABEL. See *label*.

VERSION-NUMBER. The 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 that identifies particular versions of one or more elements. See also *version selection*, *scope*, and *configuration specification*.

VERSION TREE. The hierarchical structure in which all the *versions* of an *element* are (logically) organized. The version tree display also shows *merge* operations.

VIEW. A ClearCase object that provides a work area for one or more users. 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 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 PATHNAME. A pathname that begins with a view prefix (for example, **/view/alpha** on UNIX, or **M:\alpha** on Windows) and specifies the 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 host, that records errors in accessing the view storage areas on that host.

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.

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.

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

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 used 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*.

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*.

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

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 *metadata* and VOB objects are stored. This area is managed by the database management software embedded in ClearCase. 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 *metadata* (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 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 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. **1)** An object stored in a *VOB.* **2)** 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 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 metadata.

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, 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, 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 (\). 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.

WINK IN. **1)** To cause 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*. **2)** To convert a nonshareable derived object to a shared derived object.

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.*