Rational Software Corporation®

RATIONAL® CLEARCASE MULTISITE®

ADMINISTRATOR'S GUIDE

UNIX/WINDOWS EDITION

VERSION: 2002.05.00 AND LATER

PART NUMBER: 800-025073-000

Rational®
the software development company

Administrator's Guide
Document Number 800-025073-000 October 2001
Rational Software Corporation 20 Maguire Road Lexington, Massachusetts 02421

IMPORTANT NOTICE

Copyright

Copyright © 1992, 2001 Rational Software Corporation. All rights reserved.
Copyright 1989, 1991 The Regents of the University of California
Copyright 1984–1991 by Raima Corporation

Permitted Usage

THIS DOCUMENT IS PROTECTED BY COPYRIGHT AND CONTAINS INFORMATION PROPRIETARY TO RATIONAL. ANY COPYING, ADAPTATION, DISTRIBUTION, OR PUBLIC DISPLAY OF THIS DOCUMENT WITHOUT THE EXPRESS WRITTEN CONSENT OF RATIONAL IS STRICTLY PROHIBITED. THE RECEIPT OR POSSESSION OF THIS DOCUMENT DOES NOT CONVEY ANY RIGHTS TO REPRODUCE OR DISTRIBUTE ITS CONTENTS, OR TO MANUFACTURE, USE, OR SELL ANYTHING THAT IT MAY DESCRIBE, IN WHOLE OR IN PART, WITHOUT THE SPECIFIC WRITTEN CONSENT OF RATIONAL.

Trademarks

Rational, Rational Software Corporation, the Rational logo, Rational the e-development company, Rational Suite ContentStudio, ClearCase, ClearCase MultiSite ClearQuest, Object Testing, Object-Oriented Recording, Objectory, PerformanceStudio, PureCoverage, PureDDTS, PureLink, Purify, Purify'd, Quantify, Rational Apex, Rational CRC, Rational PerformanceArchitect, Rational Rose, Rational Suite, Rational Summit, Rational Unified Process, Rational Visual Test, Requisite, RequisitePro, RUP, SiteCheck, SoDA, TestFactory, TestMate, TestStudio, The Rational Watch, among others are trademarks or registered trademarks of Rational Software Corporation in the United States and in other countries. All other names are used for identification purposes only, and are trademarks or registered trademarks of their respective companies.

Sun, Solaris, and Java are trademarks or registered trademarks of Sun Microsystems, Inc.

Microsoft, the Microsoft logo, the Microsoft Internet Explorer logo, Windows, the Windows logo, Windows NT, the Windows Start logo are trademarks or registered trademarks of Microsoft Corporation in the United States and other countries.

Patent

U.S. Patent Nos. 5,574,898 and 5,649,200 and 5,675,802. Additional patents pending.

Government Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational License Agreement and in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct 1988), FAR 12.212(a) 1995, FAR 52.227-19, or FAR 52.227-14, as applicable.

Warranty Disclaimer

This document and its associated software may be used as stated in the underlying license agreement. Rational Software Corporation expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability or fitness for a particular purpose or arising from a course of dealing, usage, or trade practice.

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

Preface xvii
About This Manual xvii
ClearCase Documentation Roadmap xviii
Typographical Conventions xix
Online Documentation xx
Technical Support xx

MultiSite Overview

1. Introduction to MultiSite 1
1.1 VOBs and VOB Replicas 1
Replica Names, Replica Objects, and Host Assignments 2
Differences Among Sites 2
Element Ownership and Ownership Preservation 4
Requirements for Ownership-Preserving Replicas 5
Synchronizing Replicas in a VOB Family 6
MultiSite, Time, and Time Zones 6
1.2 Enabling Independent VOB Development: Mastership 7
Replica Mastership 8
Branch Mastership 8
Creation of the main Branch of an Element 10
Synchronizing Development on Different Branches 10
Default and Explicit Branch Mastership 13
Type Object Mastership 14
Mastership Restrictions 17
1.3 Supporting Serial Development in Replicas 20
1.4 Conflict Resolution 21
Resolving Conflicts Among Type Objects 21
1.5 VOB Objects and Replica Objects 23

1.6 VOB Operations and the Oplog 24
Tracking Operations for Each Replica 25
Epoch Numbers 26
Optimization and the Epoch Number Matrix 27
Indirect Synchronization 29

2. Planning a MultiSite Implementation 33
2.1 MultiSite Installation 34
2.2 MultiSite Licensing 35
2.3 ClearCase Use Model 35
Branching and Mastership 36
Use of Metadata 37
Text Mode for Replicas 37
Use of Administrative VOBs or UCM 38
2.4 MultiSite Use Model 38
Type of Administration 38
Mastership Strategy 40
Replica Permission Strategy 40
Synchronization Method 41
Synchronization Pattern 42
Directions of Exchange 42
One-to-One and Ring Synchronization 43
One-to-Many Synchronization 44
Many-to-Many Synchronization 46
Synchronization Schedule 46
Use of MultiSite for Backups 47
Scrubbing Parameters for VOB Replicas 47
Oplog Scrubbing 48
export_sync Scrubbing 49
2.5 Responsibilities of MultiSite Administrators 49

3. MultiSite Command Set 53
3.1 Location of MultiSite Programs 53
3.2 multitool Use 54
multitool Subcommands 55
Commands Copied from ClearCase 55
Replica Creation, Synchronization, and Management 55
Object Mastership 56
Failure Recovery 57
3.3 View Contexts and VOB Mounts 58
3.4 Specifying VOBs and Replicas in Commands 58
3.5 Additional MultiSite Commands 59
3.6 ClearCase Commands Related to MultiSite 60

Using MultiSite

4. Creating Replicas 65
4.1 Overview of Replica Creation 65
4.2 Timing of Replica Creation 66
4.3 Notes on Different Transport Methods 66
Store-and-Forward Method 66
Communication Between Replica Hosts 66
Limiting the Size of a Packet 67
Transport Options 67
Notes on Using Tape or a File-Based Transfer Method 67
4.4 Replica-Creation Scenario 68
Planning the Rules of the Road 68
Prerequisites 70
Export Phase 71
Transport Phase 73
Import Phase 73
4.5 Replicating a VOB Between UNIX and Windows 77

5. ClearCase Feature Levels 79
5.1 Overview of Feature Levels 79
5.2 Raising the Replica Feature Level 80
5.3 Raising the VOB Family Feature Level 82
VOB Families with Bidirectional Synchronization 82
VOB Families with Unidirectional Synchronization 82
5.4 Displaying Feature Levels 83
5.5 Feature Levels Error Message 84

6. Synchronizing Replicas 85
6.1 Synchronization Patterns 86
Designing an Update Strategy 87
6.2 Assumption of Successful Synchronization 90
6.3 Transferring Packets with Store-and-Forward 90
Packet Storage During the Export and Import Phases 91
Packet Transport 92
Configuring the Store-and-Forward Facility 92
Submitting Packets to Store-and-Forward 92
Sending Files That Are Not Packets 93
Setting Up an Indirect Shipping Route 93
Retries, Expirations, and Returned Data 94
Error Notification in a Mixed Environment 95
Differentiating Packets with Storage Classes 95
6.4 Using MultiSite through a Firewall 96
Using Electronic Mail 96
Using FTP 97
Using Custom Software 98
Installing Store-and-Forward on a UNIX Firewall Host 98
Firewall Issues 100
Installing shipping_server on a Firewall 100
Controlling Ports Used by albd_server and shipping_server 101
Guidelines for Setting Port Values 101
Specifying Port Values 101

6.5 Manual Synchronization 102
Export Phase 102
Transport Phase 103
Import Phase 103
6.6 Automated Synchronization 104
Using the ClearCase Scheduler 104
Export Phase 105
Transport Phase 106
Import Phase 107
Defining Receipt Handlers 108
Scheduling Import Jobs 108
6.7 Listing Synchronization History 109
6.8 Synchronizing More Efficiently 109
Example of Increased Efficiency 109
Example of Decreased Efficiency 110

7. Managing Replicas 111
7.1 Displaying Properties of a Replica 111
7.2 Listing the Synchronization History of a Replica 113
7.3 Changing the Host Name for a Replica 113
7.4 Changing Ownership Preservation 114
7.5 Setting the Connectivity Property 116
7.6 Renaming a Replica 116
7.7 Moving a Replica 117
7.8 Changing Mastership of a Replica 117
7.9 Deleting a Replica 119

8. Managing Mastership 121
8.1 Listing an Object's Master Replica 122
8.2 Listing Objects Mastered by a Replica 123
8.3 Listing the History of Mastership Changes for an Object 123
8.4 Displaying Mastership Request Settings 124
8.5 Assigning Branch Mastership During Element Creation 124

8.6 Changing Mastership 126
Transferring Mastership of a Type Object 127
Transferring Mastership of a Replica Object 128
Transferring Mastership of a VOB 130
Transferring Mastership of an Element 131
Transferring Mastership of a Branch 132
Transferring Branch Mastership 132
Removing Explicit Mastership of a Branch 133
Transferring Mastership of a Stream 135
Transferring Mastership of All Objects Mastered by a Replica 135
Fixing an Accidental Mastership Change 137
8.7 Working with Type Objects 137
Creating a Shared Type Object 137
Listing Whether a Type Object Is Shared or Unshared 138
Converting an Unshared Type Object to a Shared Type Object 138

9. Implementing Requests for Mastership 141
9.1 Overview of a Request for Mastership 141
9.2 Requirements and Recommendations 144
9.3 Planning Your Implementation 145
To Hide Request for Mastership Features 145
9.4 Enabling Requests for Mastership 146
Prerequisites 146
Adding Developers to the Access Control List 146
Deny Requests for Specific Objects 148
Enable Requests at the Replica Level 148
9.5 Customizing Synchronization Updates for Mastership Requests 149
9.6 Displaying Mastership Request Settings 150
9.7 Troubleshooting 151
Troubleshooting Commands 151
Status Messages 152
9.8 Serial Development Scenario 157
Planning the Implementation 157
Setting Up Access Controls 157

Writing Config Specs 159
Boston 159
San Francisco 159
Tokyo 159
Requesting Mastership 160
Serial Development of a File That Cannot Be Merged 160
Serial Development of a File That Can Be Merged 161

10. Troubleshooting MultiSite Operations 163
10.1 Troubleshooting Tips 164
10.2 Replica-Creation Problems 165
Export Phase 165
Import Phase 166
Conflict in VOB Object Registry 166
Conflict in VOB-Tag Registry 167
10.3 Synchronization Export Problems 167
Cannot Find Oplog 168
Sites Have IP Connection 168
Sites Do Not Have IP Connection 169
Oplog Gap Detected During Creation of Update Packet 170
Export Failure During Version Construction 170
Packets Accumulate in Outgoing Storage Bay 171
Replica Cannot Update Itself 171
10.4 Transport Problems 172
Error Messages 172
Invalid Destination 173
Delivery Fails 174
Shipping Server Fails to Start or Connection Is Refused 174
Shipping Order Expires 175
10.5 Synchronization Import Problems 175
Packets Accumulate in Incoming Storage Bay 176
Packet is Not Applicable to Any Local VOB Replicas 177
Read from Input Stream Fails 178

Element Changes During Operation 178
rmreplica Operation Cannot be Imported 179
Replica Incarnation is Old 180
Miscellaneous Problems 181
Recovering from Lost Packets 182
Lost Replica-Creation Packet 182
Lost Update Packet 182
Inconsistent Changes to Replica 184
Ownership Preservation 185
Object Mastership 186
Automatic Renaming of Type Objects and Replica Objects 187
10.6 Running epoch_watchdog 188
10.7 Restoring and Replacing Replicas 190
Restoring a Replica from Backup 191
Replacing an Existing Replica 192
Saving Views from the Replaced Replica 195
10.8 Cleaning Up from Accidental Deletion of a Replica 196

Using MultiSite for Backup and Interoperability

11. Backing Up VOBs with MultiSite 199
11.1 Using a Backup Replica 199
Handling Objects That Are Not Replicated 200
Designing Synchronization Strategy 200
11.2 Using Replicas with Incremental Backup 201
11.3 Restoring a Replica from Backup 201

12. Using MultiSite for Interoperability 203
12.1 Advantages and Disadvantages 203
12.2 Restrictions on Multiple Replicas in a LAN 204
12.3 Setting Up Multiple Replicas at One Site 205

MultiSite Reference Pages

13. MultiSite Reference Pages 209
apropos 211
chepoch 213
chmaster 217
chreplica 224
epoch_watchdog 227
lsepoch 229
lsmaster 232
lspacket 237
lsreplica 240
mkorder 244
mkreplica 249
MultiSite Control Panel 263
multitool 268
recoverpacket 272
reqmaster 276
restorereplica 283
rmreplica 287
shipping.conf 289
shipping_server 294
sync_export_list 297
sync_receive 305
syncreplica 310

Index 321

Figures

Figure 1 Branch Mastership 9
Figure 2 Creating an Instance of a Type 16
Figure 3 Resolving Conflicts in Names of Type Objects 22
Figure 4 History of Changes to an Unreplicated VOB 25
Figure 5 State of a VOB Family 25
Figure 6 State of a Replicated VOB 26
Figure 7 Updates Between Two Replicas 27
Figure 8 Two-Row Epoch Number Matrix at Replica boston_hub 28
Figure 9 Epoch Number Matrix at Replica boston_hub 30
Figure 10 Unidirectional and Bidirectional Updating 42
Figure 11 One-to-One Synchronization Pattern 43
Figure 12 Ring Synchronization Pattern 43
Figure 13 Single-Hub Synchronization Pattern 44
Figure 14 Multiple-Hub Synchronization Pattern 44
Figure 15 Tree Synchronization Pattern 45
Figure 16 Many-to-Many Synchronization Pattern 46
Figure 17 VOB Family Information 51
Figure 18 VOB Family Feature Levels 80
Figure 19 Replica Synchronization 86
Figure 20 Peer-to-Peer Synchronization Pattern 87
Figure 21 Hierarchical Synchronization Pattern 87
Figure 22 A Synchronization Export Schedule 89
Figure 23 The Store-and-Forward Facility 91
Figure 24 Store-and-Forward Configuration 99
Figure 25 Partial Synchronization Export Pattern and Schedule for Three Replicas 110

Tables

Table 1 Data Propagated Among Replicas 3
Table 2 Mastership Restrictions for VOB Objects 17
Table 3 Disk Space Needed for Storage Bay 34
Table 4 Choosing a Packet Transfer Method 41
Table 5 multitool Subcommands Copied from ClearCase 55
Table 6 Replica Creation, Synchronization, and Management Commands 56
Table 7 Object Mastership Commands 56
Table 8 Failure-Recovery Commands 57
Table 9 Additional MultiSite Commands 59
Table 10 ClearCase Commands Related to MultiSite 60
Table 11 Import Methods 107
Table 12 Error Messages from Mastership Request Management Operations 153
Table 13 Error Messages from Mastership Requests 155
Table 14 Shipping Error Messages 172

Preface

Rational ClearCase MultiSite (abbreviated to “MultiSite” in this manual) is a layered product option to Rational ClearCase. It supports parallel software development and software reuse across project teams distributed geographically and provides automated, error-free replication of versioned object bases (VOBs) and transparent access to all software elements. You can also use MultiSite as an interoperation solution in a mixed UNIX and Windows network, or as a backup mechanism.

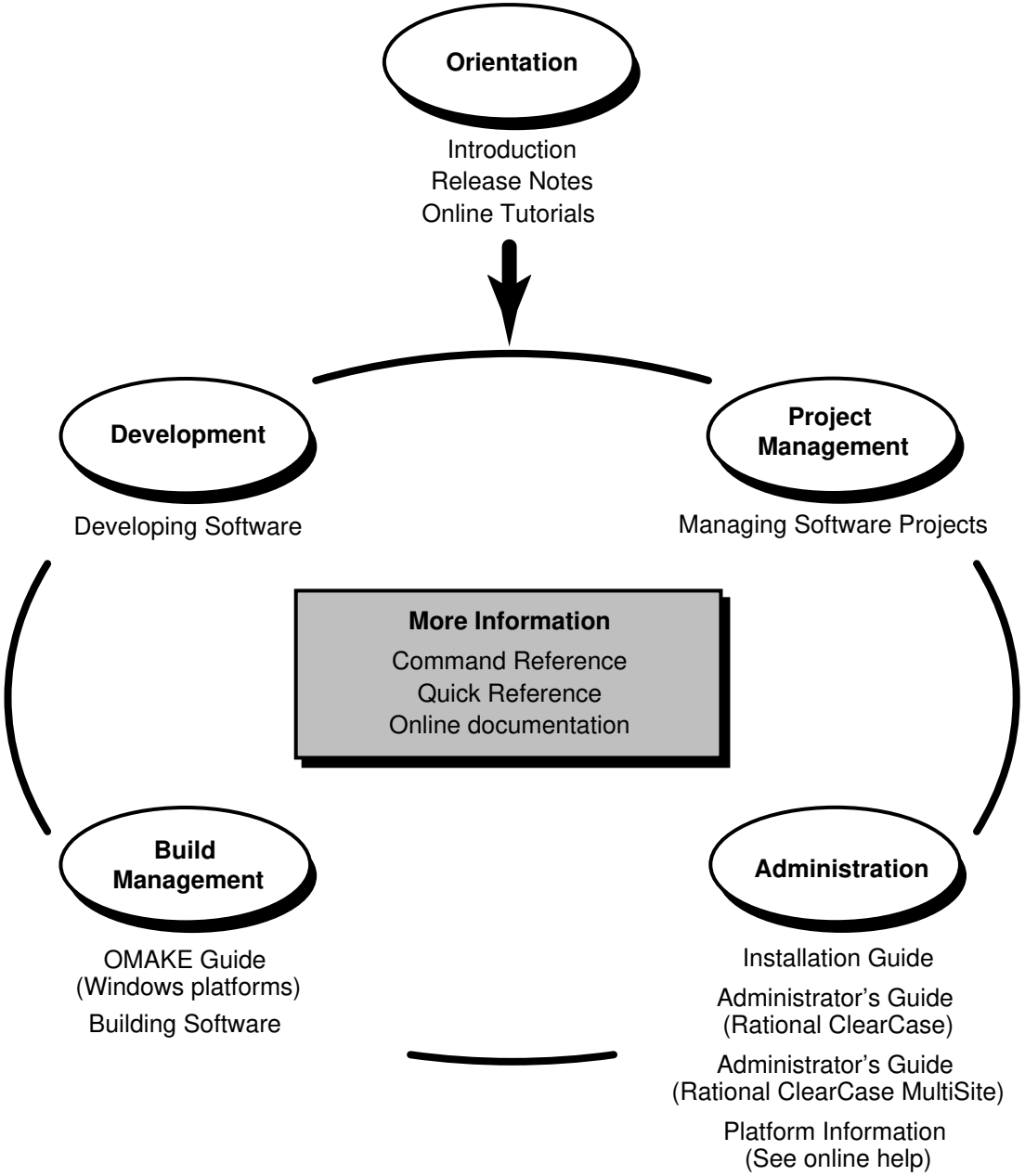
About This Manual

This manual is for all MultiSite administrators. It assumes you have experience with ClearCase. The manual provides an overview of MultiSite, describes how to set up and use it, and gives troubleshooting suggestions.

The recommended sequence for reading this manual:

- Read Chapter 1 and Chapter 3 of this book for an overview of the product.
- Read Chapter 2 to understand MultiSite planning issues.
- Read the chapters in the Using MultiSite part of the book.
- Read Chapter 11 if you plan to use MultiSite as a backup strategy.
- Read Chapter 12 if you plan to use MultiSite for UNIX and Windows interoperation.

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

MultiSite provides a help system that includes an online version of this manual and context-sensitive help for the MultiSite Control Panel and the MultiSite graphical interfaces on Windows.

MultiSite provides access to reference pages (detailed descriptions of MultiSite commands, utilities, and data structures) with the **multitool man** command.

The **multitool help** command displays individual subcommand syntax:

multitool help lspacket

```
Usage: lspacket [-long | -short] [pname ...]
```

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**

MultiSite Overview

1 Introduction to MultiSite

Rational ClearCase MultiSite adds a powerful capability to Rational ClearCase. With MultiSite, developers at different locations can use the same *versioned object base* (VOB). Each location (*site*) has a copy (*replica*) of the VOB. At any time, a site can propagate the changes made in its particular replica to other sites, by sending update packets. The update process can be automatic or can be started manually with a command.

An organization can use MultiSite to distribute independent, but related development efforts across multiple cities, nations, or continents. For example, a company in the United States has development and testing sites in India, Argentina, Japan, and Australia. Because it is impractical for all engineers to access the ClearCase VOBs in the United States, the company uses MultiSite to distribute the development.

MultiSite can also be used at a single geographical location to allow independent groups to work with the same development data, to enable interoperation in a mixed environment, or to be a backup mechanism. For example, a company that is moving some development to Windows from UNIX can create replicas on Windows instead of accessing UNIX VOBs from Windows.

1.1 VOBs and VOB Replicas

In ClearCase, a VOB provides permanent storage for an entire directory tree: directories, files, and links. The historical versions of the files in the VOB are stored in *data container* files in *storage pool* directories. The *VOB database* records the evolution of the version-controlled file-system objects, and stores the associated *metadata*, including version labels, hyperlinks, configuration records, and so on. For more details on VOB data structures, see the ClearCase documentation set.

If MultiSite is not used, each VOB has a single set of data containers and a single database. With MultiSite, some or all VOBs are replicated. A *replicated VOB* is located at multiple sites; at each site is a copy of the VOB, called a *VOB replica.* Collectively, the set of replicas of a VOB is called a *VOB family*. Each replica includes a full set of data containers and a complete copy of the VOB database. At its site, a replica appears to be a regular VOB; developers can check out, edit, and check in; build software; attach metadata annotations to objects; and so on. Regular ClearCase use models apply to use of replicas, but there are some site coordination issues that administrators must consider. (For more information, see Chapter 2, *Planning a MultiSite Implementation*.) Also, MultiSite features support simultaneous development at different replicas without conflicts. *Enabling Independent VOB Development: Mastership* on page 7 describes how conflict avoidance works.

For more details on VOBs and VOB replicas, see *VOB Objects and Replica Objects* on page 23 and *ClearCase Commands Related to MultiSite* on page 60.

Replica Names, Replica Objects, and Host Assignments

Each replica has a replica name in addition to a VOB-tag. You specify both the replica name and the VOB-tag when you create the replica. For each replica, the VOB database contains a replica object that records the name of the replica. The VOB database also tracks the location of each replica by host name. This tracking enables MultiSite administrators to specify replicas at other sites with short, mnemonic identifiers, without needing to know their exact locations.

Differences Among Sites

Each replica is a copy of the VOB, including both file-system data (data containers) and metadata (VOB database). A developer at any site can see all VOB elements, and all versions of each element.

The replicas are not necessarily exact copies of each other. MultiSite features accommodate typical differences among sites:

- Different sites may have different user spaces defined by the local password and group databases. You can configure particular replicas to ignore permissions differences, or to propagate changes in permissions from site to site (if the sites support the same user/group database). For more information, see *Element Ownership and Ownership Preservation* on page 4.

- Disk configurations and capacities may vary. Accordingly, you can manage VOB storage pools independently at each site.
- Different sites may have different development policies and can use site-specific scripts to enforce these policies. For this reason, ClearCase *triggers* are not propagated among sites.

Most, but not all, of the information stored in a VOB is replicated. All changes that create new data, remove old data, and move or rename existing data are propagated among the replicas in the VOB family.

Information stored in views is not propagated. For example, a replica update includes the fact that an element has been checked out, because the checkout is recorded in the VOB; but the update does not include the contents of the checked-out version.

Table 1 shows the information that is and is not propagated among replicas.

Table 1 Data Propagated Among Replicas (Part 1 of 2)

Data propagated	**Data not propagated**
Elements, branches, versions (including DO versions).	Derived objects that have not been checked in as versions. DOs tend to be large and short-lived; transmitting them among multiple sites is likely to be less efficient than rebuilding them at each site.
Most kinds of *type objects*.	*Trigger* type objects.
Metadata annotations: version labels, attributes, hyperlinks (including merge arrows and hyperlinks to administrative VOBs).	Individual "attached" triggers.
ClearCase UCM objects: activities, baselines, components, folders, projects, streams	
Permanent locks (those created with the **–obsolete** option).	Temporary locks (those created without the **–obsolete** option).
Checkout records of elements and changes in checked-out directories. NOTE: The **lscheckout –areplicas** command lists checkouts in other replicas.	Contents of checked-out versions.

Table 1 Data Propagated Among Replicas (Part 2 of 2)

Data propagated	Data not propagated
Event records.	
Mastership information. (See *Enabling Independent VOB Development: Mastership* on page 7.)	Mastership request settings. (See Chapter 9, *Implementing Requests for Mastership*.)
	Custom type managers.
	Changes to text mode property. (When you create a new replica, it has the same text mode property as its parent replica, but subsequent changes are not propagated.)

The biggest difference among sites reflects the basic capability of MultiSite: enabling development work to proceed independently at different locations. For more information, see *Enabling Independent VOB Development: Mastership* on page 7.

Element Ownership and Ownership Preservation

You can designate some or all replicas in a VOB family to be *ownership-preserving*. These replicas maintain the same user and group ownerships and permissions on elements, and changes to ownership or permissions are synchronized among them. The ownership of the original VOB is not preserved; the user who enters the **mkreplica –import** command becomes the owner of the new VOB.

Each replica that is not ownership-preserving maintains its own ownership and permissions information for elements. For non-ownership-preserving replicas:

- The user who enters the **mkreplica –import** command becomes the owner of the new VOB and of all elements in it.
- This user's primary group is the group for all elements.
- The initial permissions of the elements are the same as their values in the replica at which the **mkreplica –export** command is entered.

- Changes to ownership and permissions are not propagated to other replicas. Ownership and permissions changes made at ownership-preserving replicas are ignored at non-ownership-preserving replicas.

Requirements for Ownership-Preserving Replicas

The sites of ownership-preserving replicas must support the same set of user and group accounts (at least for the accounts that can be assigned to VOB elements). The user and group names and numerical IDs must be the same across sites. For example, on UNIX, the sites must share the same NIS map. On Windows, the replicas must be in the same Windows domain.

On UNIX, you can maintain separate but identical user/group databases across domains. On Windows, ownership modes (*UID*s and *GID*s) are not consistent across domains.

Therefore, the entire set of replicas cannot be ownership-preserving in either of the following cases:

- All replicas in a VOB family are not in the same Windows domain.
- Some replicas in a VOB family are located on UNIX machines, and others are located on Windows machines.

You can maintain ownership preservation on a subset of replicas in a VOB family. For example:

- A VOB family consists of the replicas **bangalore** and **tokyo**, hosted on Windows, and the replicas **boston_hub**, **sanfran_hub**, **buenosaires**, and **sydney**, hosted on UNIX. The VOB hosts for **boston_hub** and **sanfran_hub** are in domains that have the same user/group databases, so **boston_hub** and **sanfran_hub** are created as ownership-preserving replicas.
- A VOB family consists of five replicas on Windows: **seattle**, **aloha**, **troy**, **boston**, and **boston_backup**. All replicas except **boston** and **boston_backup** are located in different Windows domains. The replica **boston_backup** is used as a backup replica for **boston**, and the hosts for these replicas are in the same Windows domain (but in different ClearCase registry regions). **boston** and **boston_backup** are created as ownership-preserving replicas.

NOTE: There can be only one subset of ownership-preserving replicas in a VOB family, even if some replicas do not exchange update packets with all other replicas in the family.

Synchronizing Replicas in a VOB Family

Because elements in a replicated VOB are developed concurrently at different sites, the contents of each replica in a *VOB family* tend to diverge. In fact, a particular replica is rarely—and need never be—in the same state as any other replica. To keep the replicas from diverging too much, each site sends updates to one or more other sites. Updating a replica changes both its database and its storage pools to reflect the development activity that has taken place in one or more other replicas.

Replica information is sent in packets. A *logical packet* includes all the information required to create a new VOB replica (replica-creation packet) or to update one or more existing VOB replicas (update packet). For flexibility, and to accommodate limitations of data-transport facilities, each logical packet can be created as a set of *physical packets*.

After a logical packet is sent to a site, it is processed at that site by a **mkreplica** or **syncreplica** command invoked with the **–import** option. The changes that occurred originally at the sending site (and perhaps some other sites, too) are added to the database and storage pools of the replica at the receiving site. If the logical packet includes several physical packets, the import commands always process the physical packets in the correct order. No error occurs if the same packet is imported two or more times at a site, unless the imports occur simultaneously.

You can match the synchronization strategy for each VOB to its particular use patterns, your organization's needs, and the level of connectivity among the sites. For one VOB, you can update replicas every hour, using a high-speed network; for another VOB, you can send updates only once or twice a month, using electronic mail, magnetic tape (UNIX), or disk files as the delivery mechanism. See *MultiSite Use Model* on page 38 for information about planning synchronization. Chapter 6, *Synchronizing Replicas*, discusses the user-level facilities that create and synchronize VOB replicas. *VOB Operations and the Oplog* on page 24 describes the underlying mechanism that supports the user-level facilities.

MultiSite, Time, and Time Zones

In ClearCase and MultiSite, time stamps are stored in Universal Coordinated Time (UTC) and are printed to reflect the local time. For example, if a developer in Bangalore checks in a version of a file at 14:33 Bangalore time, the version-creation time is stored as 09:03UTC. When a developer in San Francisco looks at the description of the version, the version-creation time is displayed as 01:03 San Francisco time.

When you automate synchronization, you must adjust schedules for time zone differences. For an example, see *Designing an Update Strategy* on page 87.

Time rules in config specs are not absolute. The version selected by a time rule can change after an update packet is imported at your site. For example, your config spec has the following time rule, which selects the latest version on the **main** branch as of July 10 at 7:00 P.M.:

```
element /vobs/dev/plan.txt /main/LATEST -time 10-Jul.19:00
```

When you put this rule in the config spec, the latest version on the main branch was 17. However, a developer at another site created version 18 on July 10 at 6:00 P.M. your time, but this change has not been propagated to your site. After the update packet that contains the change is imported at your site, your time rule selects version 18.

1.2 Enabling Independent VOB Development: Mastership

Because changes to the VOB are made independently at multiple replicas, these changes may conflict. This section describes the mechanism that prevents most conflicts, and *Conflict Resolution* on page 21 describes the facilities for resolving the unavoidable conflicts.

If the development work done in different replicas were truly independent, the result would be chaos. Suppose version 17 of file **project.c** is created on the main branch in three replicas at the same time. Which is the real version 17, and what ought to happen to the other two versions?

An exclusive-right-to-modify scheme, called *mastership*, avoids such conflicts. Certain VOB-database objects are assigned a *master replica* (or *master*). The initial master of an object is the replica where the object is created, and mastership can be changed subsequently (see Chapter 8, *Managing Mastership*). In general, an object can be modified or deleted only at its master replica.

For example, this command fails because it is entered at the **boston_hub** replica, for a type object mastered by the **sanfran_hub** replica:

```
SUSHI> cleartool rename lbtype:SF_V2.0 SANFRAN_V2.0
cleartool: Error: Unable to perform operation "rename type" in replica
"boston_hub" of VOB "/vobs/dev".
cleartool:Error:Master replica of label type "SF_V2.0" is "sanfran_hub".
cleartool:Error:Unable to rename type from "SF_V2.0" to "SANFRAN_V2.0".
```

Replica Mastership

When you create a new replica, its replica object (the VOB database object that records the replica's existence) is mastered by the creating replica. Therefore, you can modify or delete the replica object only at the creating replica.

To facilitate replica maintenance, we recommend that each replica be self-mastering, which means that it is the master of its own replica object. For more information, see *Transferring Mastership of a Replica Object* on page 128.

NOTE: To perform certain procedures on a replica object, you must make the replica self-mastering. This requirement is documented in those procedures.

Branch Mastership

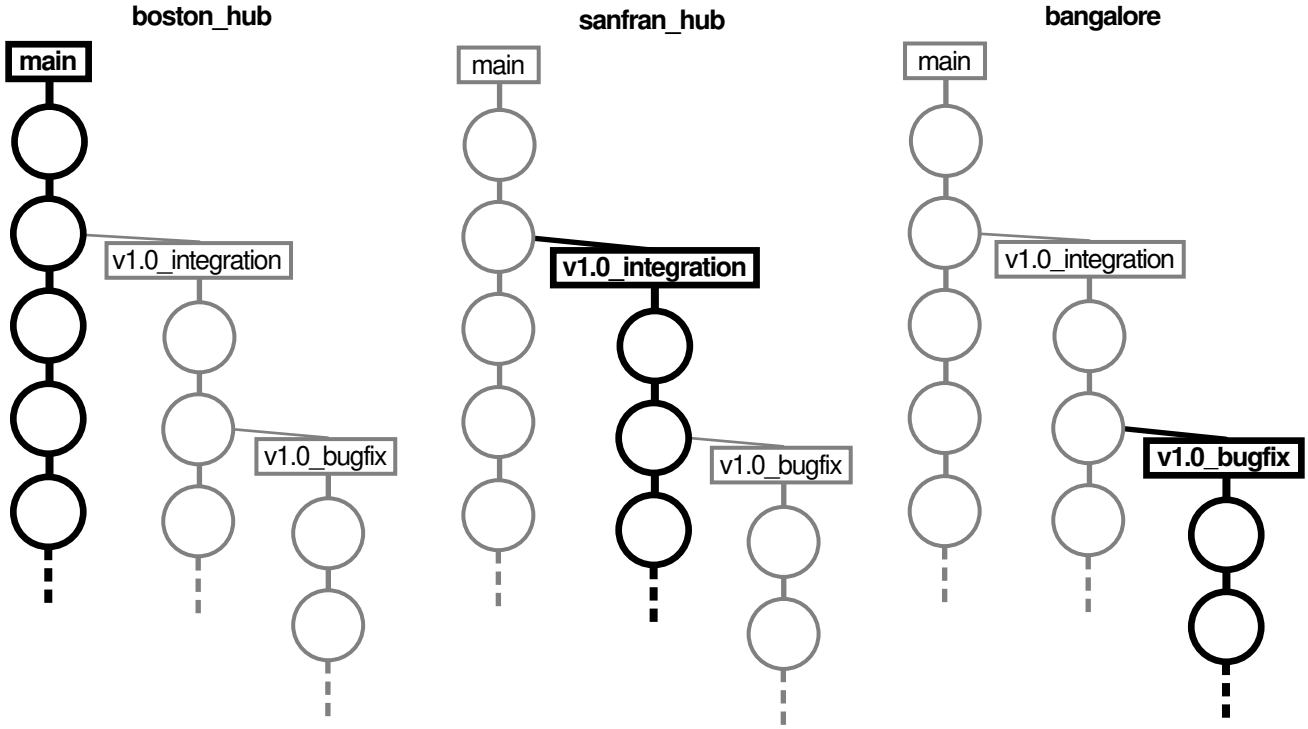
Branch mastership is the scheme that supports independent development work at different sites. By default, every branch type defined in a VOB (including the **main** branch type) is mastered by one replica in a VOB family. By default, branches can be created and modified only at the replica that masters the branch type. Checking out a version is considered a branch modification. (The exception to the creation rule is the creation of the **main** branch; for more information, see *Creation of the main Branch of an Element* on page 10.)

NOTE: Remember that a branch is an instance of a branch type. For example, **main** is a branch type, and **acc.c@@/main** and **resource.h@@\main** are branches.

The branch mastership strategy works with the standard ClearCase strategy of using branches to isolate changes for particular development tasks. (For example, fixing a defect may require changes to 5 elements, where each change is made on a branch named **v1.0_bugfix**.) With MultiSite, work on various tasks can be done at different sites, each using its own branch. The work on different branches can be propagated among sites, and then merged, as often as required by an organization's development strategy. Because the branches of an element are independent, changes made at different sites do not conflict.

Figure 1 illustrates branch mastership strategy: each replica masters a branch type and can create versions only on the branch of that type.

Figure 1 Branch Mastership



Branch mastership is implemented at both the branch type level and the branch level:

- By default, the replica in which a branch type is created masters the branch type and all instances of that branch type. For example, the **sanfran_hub** replica masters the *branch type* object named **v1.0_integration** and owns the right to modify **v1.0_integration** branches in all of the elements in the VOB.

- An administrator or developer can transfer the mastership of an individual branch (an instance of a branch type) to another replica. This feature enables serial development. For example, if a developer at the Boston site needs to work on the **v1.0_integration** branch for the element **main.c**, the San Francisco administrator can transfer mastership of the branch **main.c@@/main/v1.0_integration** to **boston_hub**, or the developer can request mastership of the branch.

  For more information on supporting serial development with MultiSite, see *Supporting Serial Development in Replicas* on page 20.

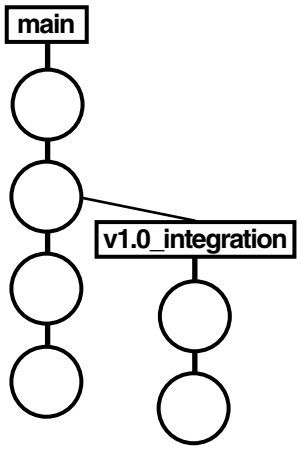
Creation of the main Branch of an Element

There is an exception to the rule that a branch can be created only at the master replica of the branch type. When you add a file to source control or create a new directory element, the **main** branch is created even if your current replica does not master the **main** branch type. By default, the **main** branch of a new element is mastered by the replica that masters the **main** branch type, and you cannot create new versions on the branch. During element creation, you can specify an option to have your current replica master all newly created branches. For more information, see *Assigning Branch Mastership During Element Creation* on page 124.

Synchronizing Development on Different Branches

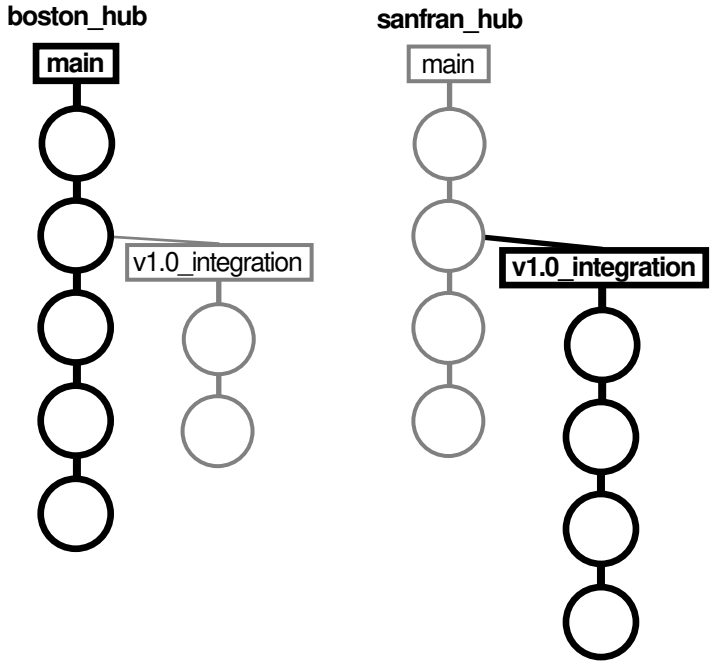
Development of an element with multiple branches can take place in different replicas concurrently, with occasional synchronizations. (The more frequently you update, the easier it is to track and reconcile the changes on different branches of elements. To reconcile changes, you use the ClearCase version-comparison and merge facilities.)

For example, before the Boston site starts using MultiSite, the element **cmdsyn.c** has two branches, **main** and **v1.0_integration**:

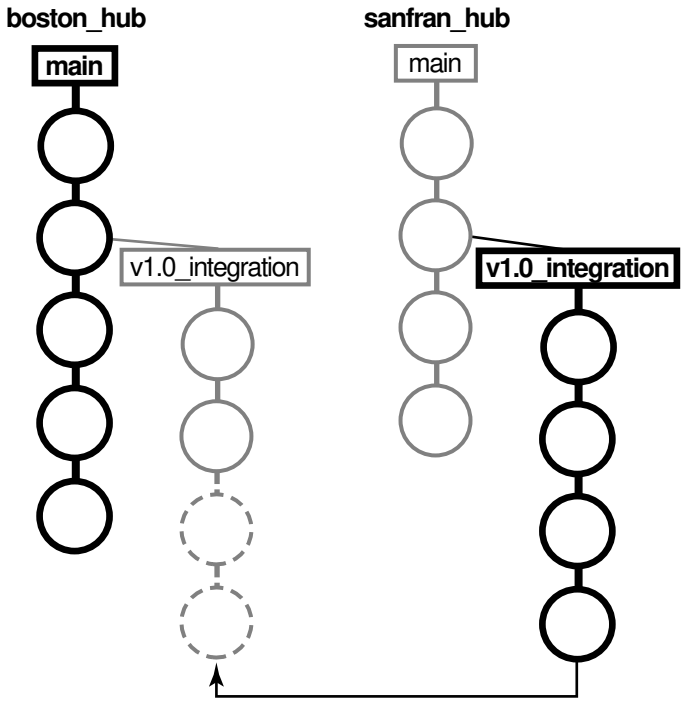


When the Boston site starts using MultiSite, the administrator creates a new replica for the San Francisco site. Because integration for Version 1.0 will be done at the San Francisco site, the **sanfran_hub** replica must master the **v1.0_integration** branch type. The administrator transfers mastership of the **v1.0_integration** branch type to the **sanfran_hub** replica.

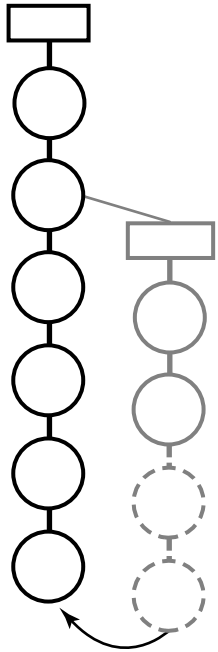
Developers in San Francisco can now create versions on the **v1.0_integration** branch of **cmdsyn.c** and can create instances of the **v1.0_integration** branch type for other elements. Work continues on the **main** branch in Boston:



The administrators at the Boston and San Francisco sites decide to merge some of the work on the **v1.0_integration** branch with the work done on the **main** branch. The San Francisco administrator sends an update packet to the **boston_hub** replica, and the Boston administrator imports it:



The Boston administrator then merges from the **v1.0_integration** branch to the **main** branch by checking out the latest version on the **main** branch, merging from the latest version on the **v1.0_integration** branch, and checking in the result of the merge:



Default and Explicit Branch Mastership

Branches can have default mastership or explicit mastership. When a branch is created, it is mastered by the replica that masters the branch type (default mastership). When you transfer mastership of a branch to another replica, that replica masters the branch explicitly. The output of **describe** shows how a branch is mastered.

For example, the branch type **v2.0_port** was created at, and is mastered by, the **sanfran_hub** replica. The **test2.txt@@/main/v2.0_port** branch has default mastership, as shown by the `(defaulted)` annotation:

multitool describe test2.txt@@/main/v2.0_port

```
branch "test2.txt@@/main/v2.0_port"
  created 18-Aug-00.10:50:34 by John Cole (jcole.user@goldengate)
  branch type: v2.0_port
  master replica: sanfran_hub@/vobs/dev (defaulted)
...
```

The administrator at the **sanfran_hub** replica transfers mastership of this branch to the **boston_hub** replica:

multitool chmaster –nc boston_hub test2.txt@@/main/v2.0_port

```
Changed mastership of branch "/vobs/dev/test2.txt@@/main/v2.0_port" to
"boston_hub"
```

The output of **describe** shows that this branch is now mastered explicitly by the **boston_hub** replica; the `(defaulted)` annotation is not present:

multitool describe test2.txt@@/main/v2.0_port

```
branch "test2.txt@@/main/v2.0_port"
  created 18-Aug-00.10:50:34 by John Cole (jcole.user@goldengate)
  branch type: v2.0_port
  master replica: boston_hub@/vobs/dev
...
```

When you transfer mastership of a branch type, mastership is transferred for all branches of that type with default mastership. Mastership of branches with explicit mastership is not transferred.

For more information, see the **chmaster** reference page and *Transferring Mastership of a Branch* on page 132.

Type Object Mastership

By default, you can create an instance of a type only in the replica that masters the type object. For example, if the **sanfran_hub** replica masters the **TESTED_BY** attribute type, you can create a **TESTED_BY** attribute only in the **sanfran_hub** replica.

Often, however, developers at different sites must create instances of the same type. For example, quality engineers at the **bangalore** replica may also use the **TESTED_BY** attribute. Therefore, the **mkattype**, **mkhltype**, and **mklbtype** commands can create two kinds of type objects:

- Instances of an *unshared type object* can be created only in its master replica. (The instances are propagated to and seen in all replicas.) Thus, there are no issues with conflicting changes made in different replicas. By default, the **mkattype**, **mkhltype**, and **mklbtype** commands create unshared type objects.
- Instances of a *shared type object* can be created in multiple replicas. To prevent cross-replica conflicts, the following restrictions apply:
  - The current replica must master the target object (the object to which the annotation is being attached).

- The ClearCase-level instance restrictions (if any) on the type object must allow creation of the instance.

NOTE: If a hyperlink type is shared, you can create a hyperlink of that type between any two objects, at any replica.

ClearCase restrictions that prevent instance creation in an unreplicated VOB also prevent instance creation in a replica; for example, if there is a lock on the type object, instance creation fails. However, because locks are not replicated (except for locks created with **–obsolete**), a lock on a shared type object in one replica does not prevent instance creation in another replica.

An unshared type object can be converted to shared, but a shared type cannot be converted to unshared. Instance restrictions (for example, *once-per-branch* use of a label type) for a shared type object cannot be changed.

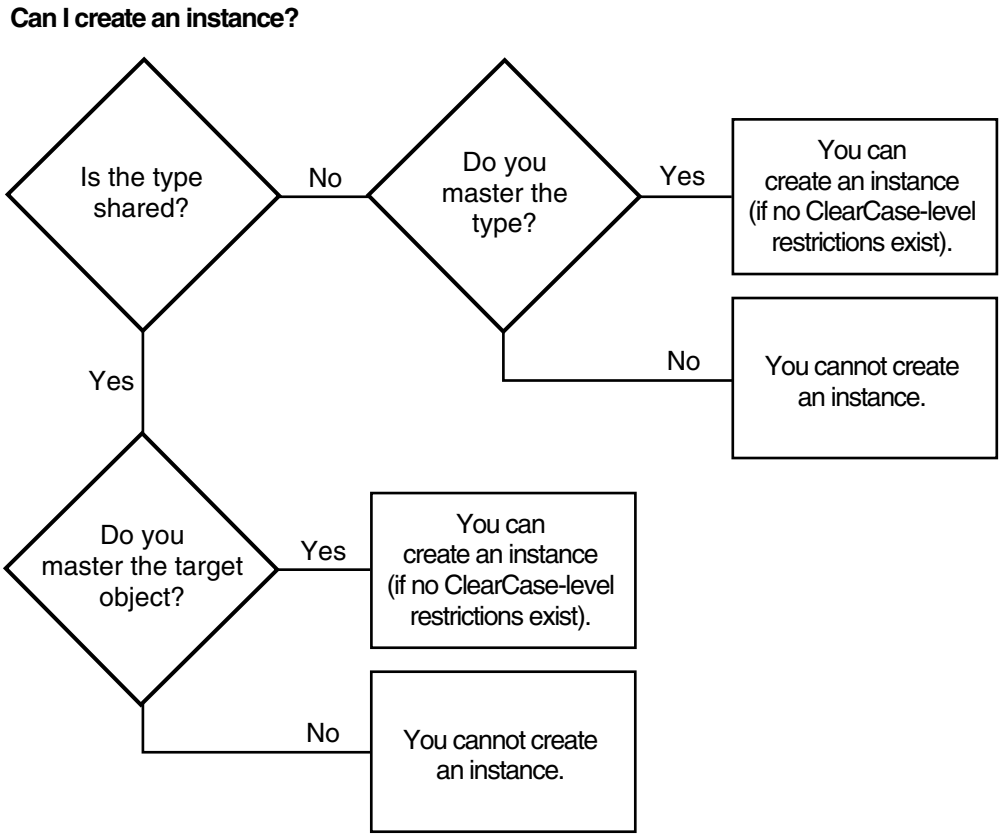
Figure 2 illustrates the restrictions on creating an instance of a shared type object. For all target objects except versions and branches, the current replica must master the target object. This is slightly different for versions and branches:

- For a version, the current replica must master the branch on which the version is located.

  NOTE: When you apply a label whose instance restriction is one per branch, your current replica must master the branch. When you apply a label whose instance restriction is one per element, your current replica must master the element.

- For a branch with default mastership, the current replica must master the branch type.
- For a branch with explicit mastership, the current replica must master the branch object.

Figure 2 Creating an Instance of a Type



For example, the administrator at **boston_hub** creates an attribute type with the following command:

cleartool mkattype –shared –vpbranch –nc TESTED

This attribute type is defined to be shared across replicas, with the restriction that at most one instance can be created on each *branch* of an element. You can create an attribute of that type on a version if both of the following are true:

- Your current replica masters that version's branch.
- No attribute of that type already exists on a version on that branch (assuming no other ClearCase restrictions).

Additional mastership restrictions exist when you use administrative VOBs and global types. If a global type is shared, ClearCase can create a local copy of the type only if the type is mastered by the administrative VOB replica at the current site. If the shared global type is not mastered at the current site, you can create instances of the type only if the client VOB replica contains a local copy of the type. This restriction applies even if your current replica masters the object to which you are attaching the instance. This mastership restriction prevents conflicting, simultaneous creation of a given type with a given name at multiple sites. For more information, see *Administrator's Guide* for Rational ClearCase.

For more information about changing type mastership, see Chapter 8, *Managing Mastership*.

Mastership Restrictions

Table 2 describes the restrictions for VOB objects.

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
Activity	Change (**chactivity**) Remove (**rmactivity**) Set (**setactivity**)	Activity
Attribute	Create (**mkattr**)	Type (if the attribute's type is unshared) Object to which attribute is being applied (if the attribute's type is shared)
	Remove (**rmattr**)	Type (if the attribute's type is unshared) Object from which attribute is being removed (if the attribute's type is shared)
Baseline	Create (**mkbl**)	Stream where you make the baseline. For an imported baseline created from a pre-UCM label, your current replica must master the component and label type.
	Label (**mklabel**)	Stream's branch type (in each VOB where you have made changes)
	Change (**chbl**) Remove (**rmbl**)	Baseline

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
Branch	Change type (**chtype**)	New branch type and the branch you are changing
	Create (**mkbranch**)	Branch type
	Remove (**rmbranch**)	Branch
Checked-out version	Reserve (**reserve**)	Branch on which the version is checked out
Component	Remove (**rmcomp**)	Component
Element	Check in (**checkin**)	Branch on which you are checking in the version
	Check out (**checkout**)	Branch on which you are checking out the version (unless you use **–unreserved –nmaster**)
	Change type (**chtype**) Relocate (**relocate**) Remove (**rmelem**)	Element
Event record	Change (**chevent**)	For a version, the branch containing the version. For any other object, the object.
Folder	Change (**chfolder**) Remove (**rmfolder**)	Folder
Hyperlink	Create (**mkhlink**)	Hyperlink type (for unshared types)
	Remove (**rmhlink**)	Hyperlink
Label	Create (**mklabel**) Remove (**rmlabel**)	If the label's type is unshared, your current replica must master the label type. If the label's type is shared, the following restrictions apply: ➤ If the label type is one per branch, your current replica must master the branch containing the version. ➤ If the label type is one per element, your current replica must master the version's element.
Merge arrow	Remove (**rmmerge**)	**Merge** hyperlink

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
Object	Change event (**chevent**) Change mastership (**chmaster**) Change name (**rename**) Lock obsolete (**lock –obsolete**) Unlock (**unlock**)	Object
	Change protection (**protect**)	Object (if current replica is ownership-preserving)
Project	Change (**chproject**) Remove (**rmproject**)	Project
Project VOB	Change list of promotion levels (**setplevel**)	**PromotionLevel** attribute type
Replica	Change host (**chreplica**) Change ownership-preservation properties (**chreplica**) Enable requests for mastership (**reqmaster**) Remove (**rmreplica**)	Replica
Stream	Change (**chstream**) Rebase (**rebase**) Remove (**rmstream**)	Stream
Symbolic link	Remove (**rmelem**)	Symbolic link
Type	Copy (**cptype**)	The replica containing the original type must master that type.
	Remove (**rmtype**) Replace (**mk******type –replace**)	Type
Version	Check in (**checkin**) Check out (**checkout**) Remove (**rmver**)	Branch With **checkout –unreserved –nmaster**, there are no mastership restrictions.

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
VOB	Change feature level (**chflevel**)	The replica to be changed must be self-mastering.
	Change protection (**protectvob**)	VOB (for ownership-preserving replicas)
	Set up snapshots (**vob_snapshot_setup**)	The replica must be self-mastering.
VOB family	Change feature level (**chflevel**)	VOB object

1.3 Supporting Serial Development in Replicas

The standard ClearCase development model is to use branches to develop software in parallel, and the standard MultiSite model is to master different branch types at different replicas. These models require you to merge changes from branch to branch.

However, sometimes sites must use serial development (for example, to make changes to elements whose versions cannot be merged). To support serial development, there are two models for changing mastership:

- Push Model

  The developer who needs to work on a branch asks the administrator at the master replica's site to transfer mastership of the branch and send an update packet containing the change.

- Pull Model

  The developer who needs to work on a branch requests mastership of the branch. This model is not enabled by default, and it requires the MultiSite administrator to enable requests and authorize developers to request mastership. However, after the setup is complete, the administrator does not need to be involved in the mastership request process.

  NOTE: The developer can also request mastership of branch types. For more information, see Chapter 9, *Implementing Requests for Mastership*.

There are two ways to use requests for mastership:

- If you cannot merge versions of the element, you must request mastership, and after your current replica receives mastership, you can perform a reserved checkout and do your work.
- If you can merge versions of the element, you can perform a *nonmastered checkout* of the element and do your work. At any time, request mastership. When your current replica receives mastership, merge your work (if required) and check in the file.

For more information about enabling requests for branch mastership, see Chapter 9, *Implementing Requests for Mastership*. For more information about the use models for requesting mastership, see *Working On a Team* in *Developing Software*.

1.4 Conflict Resolution

Mastership restrictions prevent most inconsistent changes in different VOB replicas, but some inconsistent changes are unavoidable. For example, a label type named **V3.0** can be created at two or more replicas at the same time. (The actual times can be quite different: between updates, while replicas evolve independently, a label type creation operation in one replica is logically simultaneous with all label type creations in the other replicas.)

To avoid many naming conflicts, the ClearCase and MultiSite administrators for a VOB family must create and enforce some naming and use rules for objects in VOBs. A ClearCase use model that is used consistently across sites reduces the potential for conflicts. For example, the administrators for a VOB family agree that all site-specific labels must include a site identifier, and labels that will be used at multiple sites are created only at a certain site.

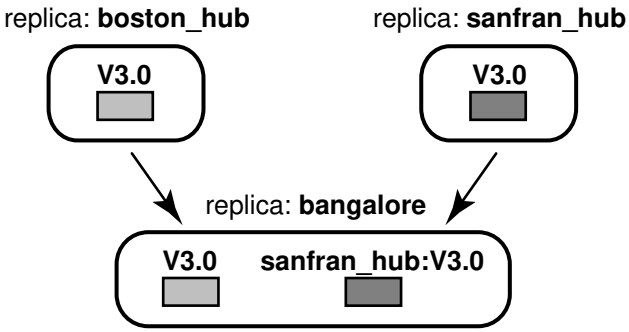
Resolving Conflicts Among Type Objects

Two objects of the same type in the same VOB cannot have identical names. Accordingly, the **syncreplica –import** command detects a conflict when an update packet includes an operation that would create a type object with the same name as an existing object at the current replica. It resolves the conflict by creating the new type object with a different name.

For example, in Figure 3, two types created at two different replicas have the same name but are different objects. When the type created at the **boston_hub** replica is imported at the **bangalore** replica, it is not renamed because the **bangalore** replica does not contain a type with that name.

However, when the type created at the **sanfran_hub** replica is imported at the **bangalore** replica, it is renamed because the **bangalore** replica already has a type with that name.

Figure 3 Resolving Conflicts in Names of Type Objects



syncreplica generates a warning message when it renames an object during import. To resolve the conflict, the Bangalore administrator must inform the Boston and San Francisco administrators of the name conflict, and they must take one of the following actions:

- Rename both label types. For example, at Boston:

  multitool rename lbtype:V2.0 V2.0_boston_hub

  At San Francisco:

  multitool rename lbtype:V2.0 V2.0_sanfran_hub

  The Boston and San Francisco administrators must then send updates to the **bangalore** replica.

- Rename one of the label types. The administrator who renames the label type sends an update to the other replicas.

For more information, see *Automatic Renaming of Type Objects and Replica Objects* on page 187.

1.5 VOB Objects and Replica Objects

It is useful to distinguish these two kinds of objects in the VOB database:

- VOB object. The database has a single VOB object. This object's UUID is listed as the `VOB family uuid` in a **lsvob –long** listing.
- VOB-replica object. The database has a VOB-replica object for each of the VOB's replicas. This object's UUID is listed as the `VOB replica uuid` in a **lsvob –long** listing.

For example:

cleartool lsvob –long /vobs/dev

```
Tag: /vobs/dev
  Global path: /net/minuteman/vobstg/dev.vbs
  Server host: minuteman
  Access: public
  Mount options:
  Region: purpledoc_unix
  Active: YES
  Vob tag replica uuid: 87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7
Vob on host: minuteman
Vob server access path: /vobstg/dev.vbs
Vob family uuid: 87f6265b.72d911d4.a5cd.00:01:80:c0:4b:e7
Vob replica uuid: 87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7
```

VOB family UUID
VOB replica UUID

Use **describe vob:** to list details about the VOB object; use **describe replica:** to list details about the VOB-replica object (the replica).

All replicas of a VOB record the same VOB object and set of VOB-replica objects. (When a new replica is created, it takes some time for the change—creation of a new VOB-replica object—to be propagated to all the replica's databases.)

1.6 VOB Operations and the Oplog

This section describes the VOB database mechanism that supports replica synchronization. This information is not required to use MultiSite, but is helpful when you want to deepen your understanding of the error-recovery facilities described in Chapter 10, *Troubleshooting MultiSite Operations*.

Most changes made to a VOB are recorded as *event records* in the VOB database. Each event record describes a change. For a replicated VOB, operation log entries (*oplogs*) are created also. These entries store all the information required to replay the changes in another replica:

- The identity of the replica where the change originally took place.
- The change to the VOB database; for example, creation of a new element, checkin of a new version, attaching of an attribute, and so on.
- The change to the storage pool, if any; for example, the contents of a new version.

  NOTE: Version information is not stored in the oplog. When version information is required by **syncreplica**, it is retrieved from the pools.
- The event record generated for the change.
- An integer sequence number: 1 for the first change originating at a particular replica, 2 for the next change, and so on. This is called the *epoch number* or *oplog-ID* of the oplog entry.

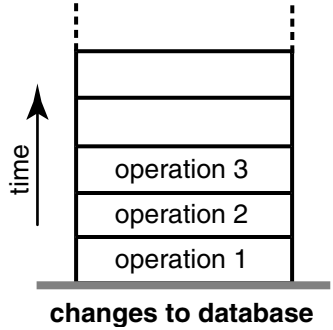
The exact kind and amount of information varies with the specific operation. For example, an oplog entry for the removal of a label has different, and less, information than an oplog entry for a **checkout** command.

NOTE: Oplog entries are created only for replicated VOBs. You can scrub a replica's oplog entries after they have been used to update other replicas. For more information, see *Scrubbing Parameters for VOB Replicas* on page 47.

Tracking Operations for Each Replica

The history of an unreplicated VOB is a linear sequence of operations (Figure 4).

Figure 4 History of Changes to an Unreplicated VOB

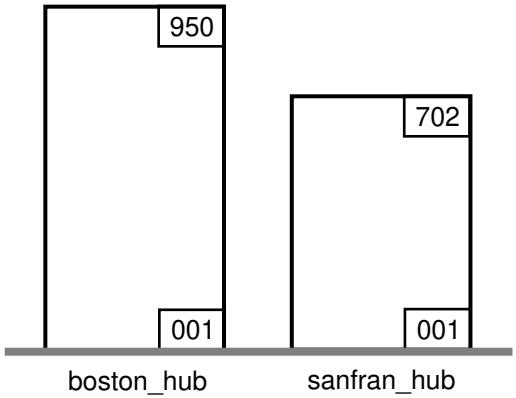


For a replicated VOB, changes are tracked separately for each replica. (That is why an oplog entry includes the identity of the replica where the operation originated.) Thus, the history of a replicated VOB can be viewed as several stacks of oplog entries. Each stack is represented by a linear sequence of epoch numbers for the operations originating in that replica.

Figure 5 shows the state of two replicas in a VOB family:

- Operations with epoch numbers 1–950 have occurred at replica **boston_hub**.
- Operations 1–702 have occurred at replica **sanfran_hub**.

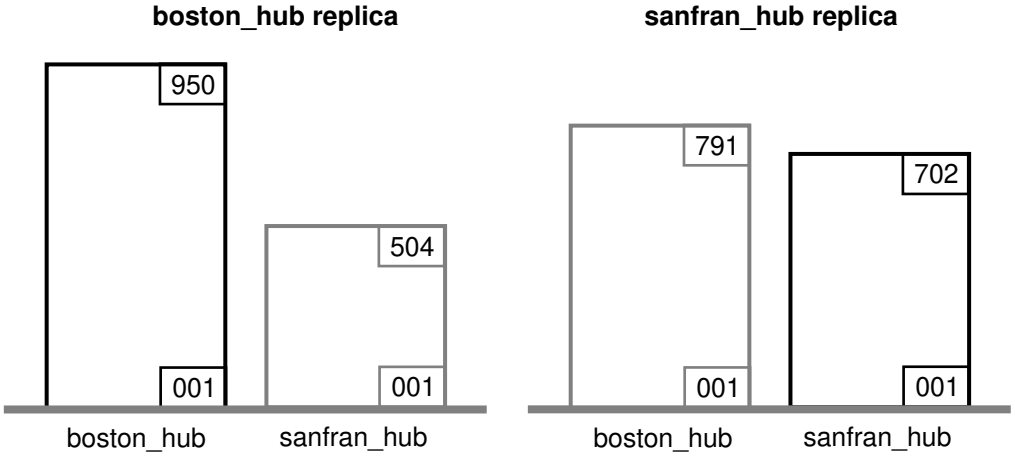
Figure 5 State of a VOB Family



A replica has accurate data only about its own operations. Until it receives update packets, its information about other replicas is out of date. For example, replica **boston_hub** records 950 local operations, but has received update packets for only 504 **sanfran_hub** operations. Similarly, replica **sanfran_hub** records 702 local operations, but has no current data about the **boston_hub** replica's state.

Figure 6 illustrates this scenario, in which each replica is out of date with respect to the operations originating at the other replica.

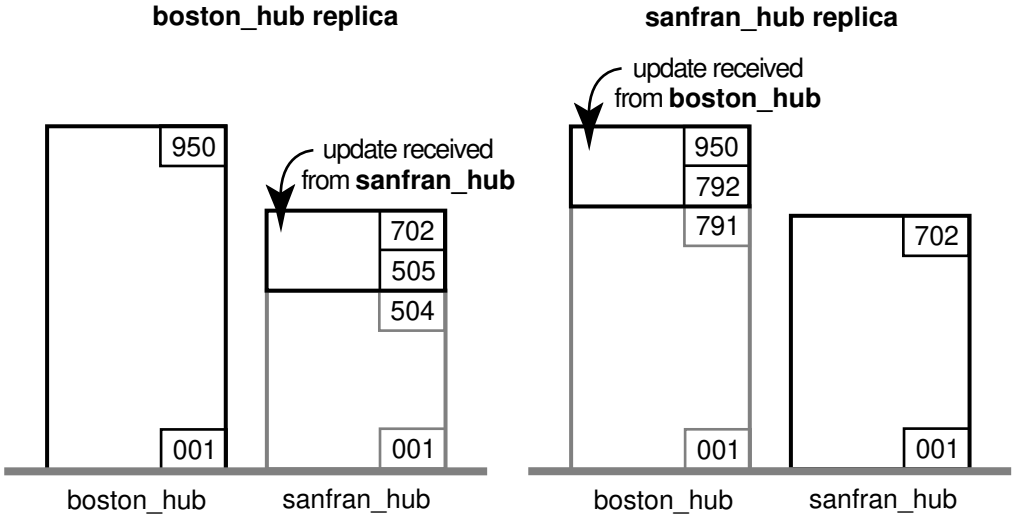
Figure 6 State of a Replicated VOB



Epoch Numbers

Picturing a replicated VOB as a set of oplog stacks, shown in Figure 6, makes it easy to understand the synchronization process. For example, an update packet sent from replica **boston_hub** to replica **sanfran_hub** consists of increments to the stack for replica **boston_hub** (operations 792–950). Figure 7 shows the two increments. Because **sanfran_hub** knows its own state, it needs updates only for other replicas. (In certain error-recovery situations, you must reset a replica's data about its own operations. See Chapter 10, *Troubleshooting MultiSite Operations*.)

Figure 7 Updates Between Two Replicas



NOTE: By the time the packet is imported at **sanfran_hub**, additional VOB-level changes may have been made at **boston_hub**. These changes are not included in the update packet.

Optimization and the Epoch Number Matrix

The MultiSite synchronization scheme attempts to minimize the amount of data transmitted among sites. Each replica keeps track of these epoch numbers:

1. **Changes made in the current replica**. The epoch number that indicates how many operations originated at the current replica.

2. **Changes at sibling replicas**. When **syncreplica** writes an operation from an update packet to the current replica, it increments the epoch number for the sibling replica at which the operation originally occurred. This epoch number is the number of operations originating at the sibling replica that have been imported at the current replica.

3. **Current knowledge of the states of other replicas**. For each other replica, an estimate of its own changes and other replicas' changes.

Figure 8 shows how these epoch numbers fall into an *epoch number matrix*. Each replica maintains its own such matrix, revising its rows as work occurs locally and as it exchanges update packets with other replicas:

- When work occurs in the **boston_hub** replica, its own number of oplog IDs is incremented.
- When the **boston_hub** replica generates an update packet to be sent to **sanfran_hub**, it revises the **sanfran_hub** row in its epoch number matrix.

  Note that a **syncreplica –export** command updates epoch numbers immediately. It does not wait for acknowledgment from the receiving site that the packet has been received and applied correctly. During normal ClearCase and MultiSite processing, no manual intervention is required to maintain the accuracy of the epoch number matrices for the various replicas. However, failure to apply a packet may require manual intervention, as described in *Lost Update Packet* on page 182.
- When the **boston_hub** replica receives an update from **sanfran_hub**, it revises its own row (**boston_hub**) and the **sanfran_hub** row in its epoch number matrix.

Figure 8 Two-Row Epoch Number Matrix at Replica boston_hub

	Operations originated at **boston_hub**	Operations originated at **sanfran_hub**
boston_hub's record of its own state	950	504
boston_hub's estimate of **sanfran_hub**'s state	912	504

The contents of this matrix are reported by the **multitool lsepoch** command at the **boston_hub** replica:

multitool lsepoch

```
For VOB replica "/vobs/dev":
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950            (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504            (sanfran_hub)
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=912            (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504            (sanfran_hub)
```

A **syncreplica –export** command entered at **boston_hub** uses this matrix as follows to generate an update destined for **sanfran_hub**:

1. At **boston_hub**, the number of local operations is 950 (number in upper-left corner of matrix), and the estimate is that **sanfran_hub** has been updated only to epoch number 912 (number in lower-left corner).

2. The update packet that **boston_hub** sends to **sanfran_hub** includes **boston_hub** oplog entries 913-950. After the Boston administrator invokes **syncreplica –export**, the **sanfran_hub** row is updated:

   multitool lsepoch

   ```
   For VOB replica "/vobs/dev":
   Oplog IDs for row "boston_hub" (@ minuteman):
    oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950        (boston_hub)
    oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504        (sanfran_hub)
   Oplog IDs for row "sanfran_hub" (@ goldengate):
    oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950        (boston_hub)
    oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504        (sanfran_hub)
   ```

Indirect Synchronization

If there are more than two replicas in a VOB family, synchronization can occur indirectly. A replica can include nonlocal changes in update packets. For example, if **boston_hub** exchanges updates with replicas **sanfran_hub** and **bangalore**, it sends **bangalore** oplog entries that it has received previously from **sanfran_hub**. These entries may or may not bring replica **bangalore** up to date on **sanfran_hub**'s changes. (An update sent from **sanfran_hub** to **bangalore** does bring **bangalore** up to date.)

NOTE: If a replica does not receive packets directly from some replicas in the VOB family, its rows for those replicas may contain zeros. This is expected behavior.

Figure 9 shows replica **boston_hub**'s epoch number matrix.

Figure 9 Epoch Number Matrix at Replica boston_hub

	Operations originated at **boston_hub**	Operations originated at **sanfran_hub**	Operations originated at **bangalore**
boston_hub's record of its own state	950	504	653
boston_hub's record of **sanfran_hub**'s state	912	504	653
boston_hub's record of **bangalore**'s state	709	221	653

The contents of this matrix are reported by the **lsepoch** command:

multitool lsepoch

```
For VOB replica "/vobs/dev":
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950            (boston_hub)
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=653            (bangalore)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504            (sanfran_hub)
Oplog IDs for row "bangalore" (@ ramohalli):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=709            (boston_hub)
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=653            (bangalore)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=221            (sanfran_hub)
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=912            (boston_hub)
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=653            (bangalore)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504            (sanfran_hub)
```

A **syncreplica –export** command at Boston uses this matrix to export an update for **bangalore**:

1. At Boston, there are 950 local operations (number in upper-left corner of matrix), and the estimate is that **bangalore** has been updated only to epoch number 709 (lower-left corner).

2. For operations that originated at **sanfran_hub**, **boston_hub** has been updated to epoch number 504, and estimates that **bangalore** has been updated only to epoch number 221.

3. The update packet that **boston_hub** sends to **bangalore** includes **boston_hub** oplogs 710-950 and **sanfran_hub** oplogs 222-504. The output of a **multitool lsepoch** command at Boston now looks like this:

multitool lsepoch

```
For VOB replica "/vobs/dev":
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950          (boston_hub)
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=653          (bangalore)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504          (sanfran_hub)
Oplog IDs for row "bangalore" (@ sushi):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950          (boston_hub)
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=653          (bangalore)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504          (sanfran_hub)
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=912          (boston_hub)
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=653          (bangalore)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=504          (sanfran_hub)
```

2 Planning a MultiSite Implementation

Before you install and use Rational ClearCase MultiSite, you need to plan your implementation. The plan should include the following items:

- MultiSite installation
- MultiSite licensing
- ClearCase use model
- MultiSite use model
- Responsibilities of MultiSite administrators

This chapter describes these issues in more detail. We recommend that you document your plan in writing and implement your design decisions in a set of test replicas before changing your development environment.

2.1 MultiSite Installation

For MultiSite installation instructions, see the *Installation Guide* for the ClearCase Product Family.

You must install MultiSite on all VOB server hosts where replicated VOBs will reside; replica creation and synchronization imports must occur on the host where the replica resides. You do not need to install MultiSite on your computer to manage mastership, because the MultiSite object mastership commands are available in **cleartool**. However, you may want to install MultiSite on your computer so you have convenient access to other MultiSite commands. You do not need to install MultiSite on ClearCase client hosts or on server hosts that will not host replicated VOBs.

Each VOB server host where replicas will reside must have enough disk space for the MultiSite *storage bay* directories. The storage bays hold MultiSite packets, along with their corresponding *shipping order* files. Table 3 describes the amount of available disk space needed on the disk partition where the storage bay is located.

Table 3 Disk Space Needed for Storage Bay

Type of packet	Disk space needed
replica-creation	Size of VOB database and VOB source pools.
update	On Windows, twice the size of the largest packet to be stored in bay. The reason is that there may be two instances of the same packet in the bay at one time: one on its way to another destination, and another waiting to be applied to the replica on the current host.
	On UNIX, the size of largest packet to be stored in bay.

There is no specific formula for determining how large your update packets will be. The general rule is that more frequent synchronization results in smaller packets. However, even if you synchronize every hour, a large amount of development activity or release activity can occur in an hour (for example, all of the executables for a release are checked in just before a build, or the release engineer labels all the versions for a release) and can cause a large packet to be generated. If you are not sure that the available disk space can accommodate an unexpectedly large packet, you can configure MultiSite to limit the size of an update packet.

For more information on specifying storage bays, see the **shipping.conf** (UNIX) and **MultiSite Control Panel** (Windows) reference pages.

2.2 MultiSite Licensing

A MultiSite license is required for any access to an object in a replicated VOB—by a MultiSite command, a ClearCase command, or a standard operating system command. You can calculate the number of MultiSite licenses your site needs by determining how many developers will access replicated VOBs. If all of your developers will access replicated VOBs, you need the same number of MultiSite licenses as ClearCase licenses. If not all developers will access replicated VOBs, you can purchase fewer MultiSite licenses.

For example, a company has two sites, with 20 developers at site A and 5 developers at site B. The company has three VOBs at site A; two of them will be replicated to site B and one will not be replicated. Five of the developers at site A will access only the unreplicated VOB, and the remaining 15 will work in all VOBs. Therefore, the company needs to purchase the following numbers of licenses:

Site	Number of ClearCase licenses	Number of MultiSite licenses
A	20	15
B	5	5

NOTE: This example assumes that you purchase a ClearCase license for each developer. If you have fewer ClearCase licenses than developers, you can purchase a proportionate number of MultiSite licenses. For example, if site B purchased three ClearCase licenses, they would also purchase three MultiSite licenses.

For more information about MultiSite license allocation and purchasing licenses, see the *Installation Guide* for the ClearCase Product Family.

2.3 ClearCase Use Model

Before development work is started in any VOB, the project manager and administrator must define the ClearCase use model. For example, the project manager must specify the branches, labels, and triggers that are used for development and integration work. The following sections describe the ways in which MultiSite use affects this planning.

Branching and Mastership

Mastership restrictions affect the choices you make about branching and merging:

- A common branching strategy is to use a single release branch (or integration branch) and multiple development branches. The project manager or developer merges changes from the development branch to the integration branch. You can use this strategy with MultiSite, but the merges to the integration branch must occur at the replica that masters the integration branch.

  You could also use a single release integration branch, multiple site integration branches, and multiple developer branches. With this scenario, developers or project managers at a replica merge to the site integration branch, and the project manager at the replica that masters the release integration branch merges to that branch from the site integration branches.

  You may need to allow developers to transfer and request mastership of branches and branch types. Developers at different sites may have to use the same branch type (for example, because an element's versions can't be merged, or because each site must merge its own work to the integration branch). A branch or branch type's mastership cannot be shared by multiple replicas; instead, there are two models for transferring mastership between sites:

  Model 1. Create a schedule that determines when each site masters the branch or branch type. Create scripts to transfer mastership.

  Model 2. Give the developers at the sites the ability to request mastership of the branch or branch type. For more information about this model, see Chapter 9, *Implementing Requests for Mastership*.

  NOTE: Do not use mastership transfer models as substitutes for good branching and merging rules. Enabling requests for mastership involves more planning and setup than implementing a strategy for branching and merging. Also, if you can develop in parallel, planned branching and merging is safer than allowing developers to request mastership and merge their own work randomly.

- You can use auto-make-branch rules in config specs only if the current replica masters the branch type in the rule. For example, if your current replica masters the **v1.0_bugfix** branch type but not the **v1.0** branch type, this config spec is incorrect because the **v1.0** branch cannot be created at this replica:

  ```
  element * CHECKEDOUT
  element * .../v1.0_bugfix/LATEST
  element * .../v1.0/LATEST -mkbranch v1.0_bugfix
  element * /main/LATEST -mkbranch v1.0
  ```

- By default, when you create an element in a replicated VOB, mastership of the branch **main** is assigned to the replica that masters the branch type **main**. If this replica is not your current replica, you cannot create new versions on the **main** branch. Also, if your config spec contains **mkbranch** rules and your current replica does not master the branch types, the branches cannot be created during element creation.

  You can assign mastership of a new element's main branch and other branches created during element creation to your current replica. For more information, see *Assigning Branch Mastership During Element Creation* on page 124.

Use of Metadata

Mastership restrictions affect the way you use ClearCase attributes, labels, or hyperlinks. You need to decide whether these types must be shared. You can create instances of an unshared type only in the replica that masters it. You can create instances of a shared type only in the replica that masters the object to which you are attaching the instance. For more information, see *Type Object Mastership* on page 14.

Trigger types and triggers are not replicated. If a trigger is in use at one replica and needs to be used at other replicas, you must send the appropriate information (for example, the output of a **describe trtype:** command and the contents of any associated scripts) to the administrators at the other sites.

Text Mode for Replicas

When you create a new replica, it has the same text mode as the replica from which it was exported. However, changes to a replica's text mode are not propagated to the other replicas in the family, so if you make a text mode change that needs to occur at all replicas in the family, you and the other MultiSite administrators must change the text mode at each replica. For more information about text modes, see the *Administrator's Guide* for Rational ClearCase.

Use of Administrative VOBs or UCM

If replicated VOBs use global types, the administrative VOBs must be replicated. For more information on global types, see the *Administrator's Guide* for Rational ClearCase.

NOTE: If a global type is shared, Rational ClearCase can create a local copy of the type only if the type is mastered by the administrative VOB replica at the current site. If the shared global type is not mastered at the current site, you can create instances of the type only if the client VOB replica contains a local copy of the type. This restriction applies even if your current replica masters the object to which you are attaching the instance. This mastership restriction prevents conflicting, simultaneous creation of a given type with a given name at multiple sites. For more information, see *Administrator's Guide* for Rational ClearCase.

If you replicate a component VOB, you must replicate its PVOB.

When you use ClearCase UCM and MultiSite, some developer and project manager tasks are different. A project's integration stream is mastered by one of the replicas in the VOB family, and developers at other replicas must do a remote deliver of their work to the stream. The project manager at the master replica completes the deliver operations. The *Developing Software* and *Managing Software Projects* manuals describe this scenario in more detail.

2.4 MultiSite Use Model

The following sections describe the different aspects of your MultiSite use model.

Type of Administration

While you are planning your implementation, you need to decide how much control the individual sites will have over their replicas. Your choices are centralized administration, individual administration, or some combination of the two.

- With centralized administration, there is a hub site. For each VOB family, all the replicas in the family are mastered by a replica at the hub site. Administrators at the hub site maintain all replicas and all synchronization patterns and schedules. These administrators have permission to access the VOB replica servers at all sites.

Advantages of this scheme:

- Your company does not have to hire a MultiSite administrator for each site.
- It is easier to make sure schedules do not conflict with each other.

Disadvantages:

- Some administrative procedures require a replica to be self-mastering.
- If ClearCase administration is done at a local level, the MultiSite administrators must have knowledge of all local administrative procedures (for example, backups and server maintenance).
- Remote access to all sites is required.

➤ With individual administration, each replica is self-mastering and there is an administrator at each site. Administrators are responsible for creating and maintaining replicas, synchronization patterns, and synchronization schedules at their sites.

Advantages of this scheme:

- No mastership changes are required when an administrator needs to change replica properties.
- Administrators can ensure that MultiSite administrative procedures do not conflict with ClearCase administration.

Disadvantages:

- A MultiSite administrator is needed at each site.
- Communication among administrators can be difficult if the company has sites in multiple time zones.

You can also have semi-centralized administration. For example, you may have MultiSite administrators at sites with major development efforts and give these administrators control over their MultiSite environment. The responsibility for administering smaller sites is distributed among the MultiSite administrators.

Mastership Strategy

The choices you make for your ClearCase use model and MultiSite administration model determine your mastership strategy. Your plan should state which replicas will master branch types, label types, elements, and other VOB objects. After you create the replicas in the VOB family, you can change mastership of objects. For more information, see *Enabling Independent VOB Development: Mastership* on page 7 and *Changing Mastership* on page 126.

Replica Permission Strategy

When you import a replica-creation packet, you must specify whether the new replica is ownership-preserving or non-ownership-preserving. In most cases, your replicas must be non-ownership-preserving. For information about the requirements for creating ownership-preserving replicas, see *Element Ownership and Ownership Preservation* on page 4.

If you plan to create one or more ownership-preserving VOB replicas, follow these steps:

1. At the exporting site, gather the current VOB ownership and group information and send it along with the packets created by **mkreplica –export**.

   a. Get the name of the VOB owner and VOB groups, using the **cleartool describe** command on the VOB object. For example:

   cleartool describe vob:/vobs/dev
   ```
   versioned object base "/vobs/dev"
     created 15-Aug-00.14:19:03 by CC Admin (ccadm.user@minuteman)
     VOB family feature level: 1
     VOB storage host:pathname "minuteman:/vobstg/dev.vbs"
     VOB storage global pathname   "/net/minuteman/vobstg/dev.vbs"
     database schema version: 53
     VOB ownership:
       owner purpledoc.com/ccadm
       group purpledoc.com/user
   ```

   b. Translate the symbolic names to numbers. On UNIX, become the VOB owner and issue the **id** command. For example:

   su ccadm
   ```
   Password: xxxxxx
   ```
   id
   ```
   uid=1083(ccadm) gid=20(user)
   ```

2. At each importing site, ensure that the user ID, primary group, and secondary groups match the information from the exporting site, in name and number.

   If they do not match, you must modify the user and group information to prevent import failures due to permissions problems, as described in *Ownership Preservation* on page 185.

   If the names are the same and the numbers are different, you must create non-ownership-preserving replicas.

Synchronization Method

There are multiple methods you can use to transport MultiSite packets. The method you choose depends on how your sites are connected, how quickly you must transfer packets, and how important security is. Table 4 lists the recommended methods for various situations.

Table 4 Choosing a Packet Transfer Method

Your situation	Recommended methods	Source of more information
Sites are connected with high-speed lines	**shipping_server**	*Transferring Packets with Store-and-Forward* on page 90 **shipping_server** reference page
One or more sites have firewalls	Tape, diskette, CD-ROM, e-mail, **ftp**, **shipping_server**	*Using MultiSite through a Firewall* on page 96 **syncreplica** reference page
Must transfer packets quickly	E-mail, **ftp**, **shipping_server**	**shipping_server** reference page, **syncreplica** reference page
No electronic connection between sites	Tape, diskette, CD-ROM	**syncreplica** reference page

Synchronization Pattern

The synchronization pattern for a VOB family defines which replicas exchange update packets and the direction of exchange. Your choice of pattern depends on the following factors:

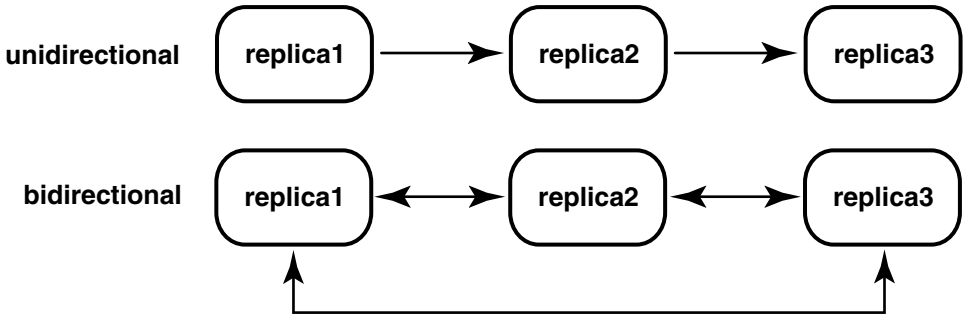
- Bandwidth between sites
- Network topology
- Latency of changes: how quickly changes made at one replica need to be received at another replica in the family
- Failure tolerance

The following sections describe unidirectional and bidirectional exchanges and the most common synchronization patterns.

Directions of Exchange

Synchronization can be unidirectional or bidirectional, as shown in Figure 10.

Figure 10 Unidirectional and Bidirectional Updating



In most cases, you will use bidirectional updating. Unidirectional updates are suitable in situations like these:

- You use a replica as a backup.
- Your company supplies source code to another site (or company) for read-only use.

- A high-security development project uses the same files as a more open project. In this case, the open project sends updates to the high-security project, but no updates are sent in the other direction.

However, unidirectional updates carry some risk. For example, an accidental change of mastership cannot be fixed, and restoring from a replica that does not exchange updates directly with the broken replica involves extra work. Also, you must ensure that no work is done accidentally in a read-only replica; do this by creating triggers or locking the VOB to prevent checkouts and creation of metadata.

One-to-One and Ring Synchronization

Figure 11 One-to-One Synchronization Pattern

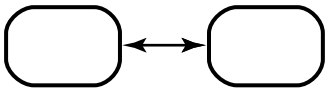
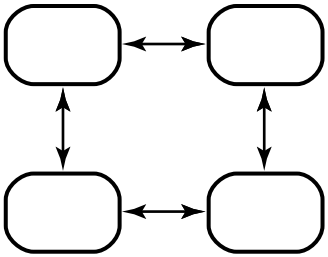


Figure 12 Ring Synchronization Pattern



The simple one-to-one and ring (or round-robin) patterns are simple patterns that are most suitable for small numbers of replicas. As the number of replicas grows larger, the amount of time increases for a change made at one replica to be received at a replica at the other side of the ring.

One-to-Many Synchronization

Figure 13 Single-Hub Synchronization Pattern

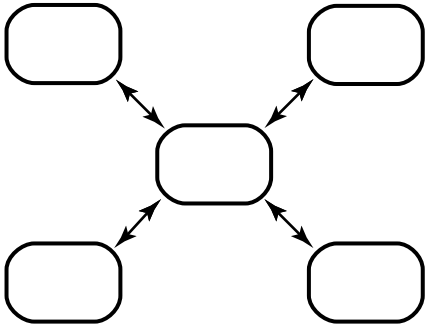


Figure 14 Multiple-Hub Synchronization Pattern

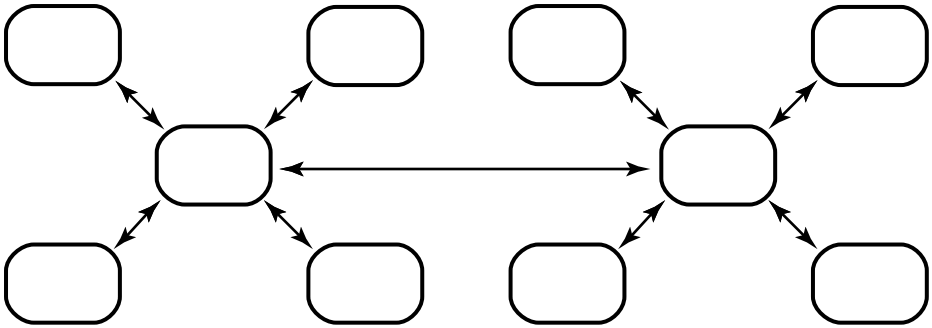
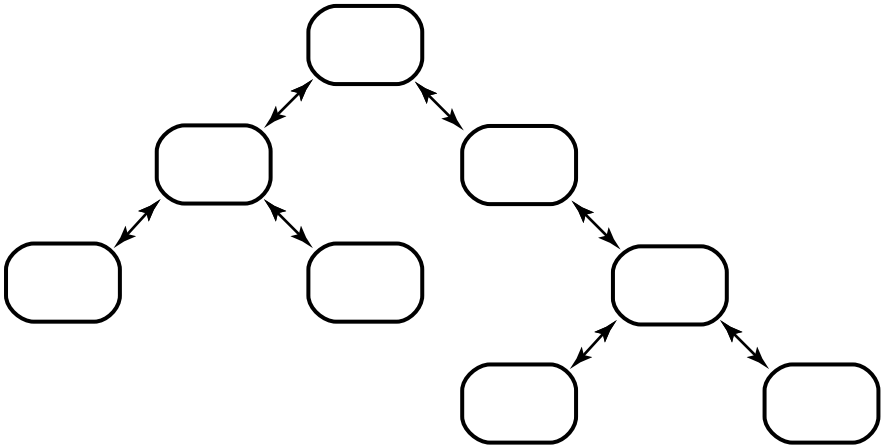


Figure 15 Tree Synchronization Pattern



Advantages:

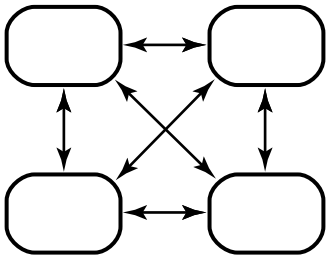
- More efficient for the spoke and branch replicas, which send to and receive from only one other replica.

Disadvantages:

- If the hub or root site goes down, all spoke/branch sites must reconfigure their pattern to keep communication going.
- If you change the synchronization pattern so that replicas that did not synchronize directly now exchange packets, the first packets that are generated may be too large for the system. To avoid this problem, you can run **chepoch –actual** regularly among the spoke or branch replicas.

Many-to-Many Synchronization

Figure 16 Many-to-Many Synchronization Pattern



Advantages:

- For companies with few sites, this pattern keeps each replica's epoch table the most accurate for all siblings.
- If one site is unavailable, the other sites do not have to change their patterns to continue synchronizing.

Disadvantages:

- Each administrator must maintain more synchronization jobs and spend more time keeping track of packets.

Synchronization Schedule

The synchronization schedule for a VOB family defines when replicas in the family send and receive updates. The schedule is affected by many factors, including the rate of development at different sites, the connections among sites, and whether you use MultiSite as a backup strategy.

Consider the following issues when planning your synchronization strategy:

- Rate of development.

  If you schedule synchronizations frequently, merging is simpler because fewer changes have taken place. Also, you lose less work if a replica is deleted accidentally and you must restore it from backup.

  Make sure that synchronizations do not overlap with VOB backups. VOBs must be locked while they are being backed up, and the **syncreplica** command fails if the VOB is locked.

- Time zone differences. Be sure to take different time zones into account when you send an update or set up automated updates. Figure 22 on page 89 illustrates synchronization taking place among replicas in three time zones.

- Use of administrative VOBs. Because local type objects in a client VOB are linked to global type objects in the administrative VOB, we recommend that you synchronize a client VOB and its administrative VOB at the same time. If you do not, users may have trouble accessing type objects.

  For example, at the Boston site, the client VOB **/vobs/dev** is linked to administrative VOB **/vobs/admin**, and both VOBs are replicated to San Francisco and Bangalore. You export update packets to replicas **sanfran_hub@/vobs/dev** and **sanfran_hub@/vobs/admin** at 11:00 P.M. local time and export update packets to replicas **bangalore@/vobs/dev** and **bangalore@/vobs/admin** at 5:00 A.M. local time. The administrator at San Francisco imports both packets at the same time, as does the administrator at Bangalore.

- Use of ClearCase UCM. We recommend that you synchronize a component VOB and its PVOB at the same time. If you do not, users may have trouble accessing baselines and activities and the versions associated with those objects.

Use of MultiSite for Backups

You can use MultiSite as part of your VOB backup strategy. For more information, see Chapter 11, *Backing Up VOBs with MultiSite*.

Scrubbing Parameters for VOB Replicas

When a ClearCase or MultiSite command makes a change to a replica, an *oplog entry* is recorded in the replica's database. (See *VOB Operations and the Oplog* on page 24 for more information on this mechanism.) Also, when you export an update packet, an export_sync record is created for each target replica. These records are stored in the VOB database and are used by the **recoverpacket** command to reset a replica's epoch number matrix.

You can scrub oplog entries and export_sync records to reclaim disk space and database records, but you must keep them long enough to ensure that you can recover from replica failures and packet losses. The following sections give guidelines for configuring scrubbing frequency.

For more information on VOB scrubbing, see the ClearCase **vob_scrubber** reference page.

Oplog Scrubbing

Oplog entries must be kept in the database for a significant period. In the near term, they are required when the replica generates update packets to be sent to all other replicas. Beyond that, entries may be required to help other replicas recover from catastrophic failures. If no replica can supply these entries, the replica being restored must be re-created. (See *Restoring a Replica from Backup* on page 191.) Because of the need to use oplog entries during synchronization, your synchronization strategy determines how often oplogs can be scrubbed.

By default, an oplog entry is never scrubbed. Do not change this setting until you establish the synchronization pattern in the VOB family and verify that packets are being exported and imported successfully.

When it is safe to delete oplog entries for a replica, follow these steps:

1. Coordinate with administrators at other sites to decide how long each site must keep oplog entries.

   Each site must keep entries for as long as necessary to allow **restorereplica** operations to complete successfully. The frequency with which you scrub oplogs depends on the following factors:

   - The pattern of synchronization among replicas in the VOB family
   - How often the replicas are synchronized

     Frequency of synchronization refers both to how often updates are exported and to how often they are imported at other sites. Also, consider setting up a verification scheme so you can ensure that packets are processed successfully at other replicas before any oplog entries are scrubbed.

   - How often you back up the replicas

     For example, if a VOB is backed up weekly at all sites and you want to be able to restore to the backup from two weeks ago, each replica must keep three weeks of oplog entries. If replicas synchronize weekly, you must assume that the weekly packet hasn't been sent to the other replica, and add another week. Finally, for extra security, add another month. The result is a scrubbing time of two months.

2. Change the oplog scrubbing parameter for your replica:

   a. Copy *ccase-home-dir*/**config/vob/vob_scrubber_params** (UNIX) or *ccase-home-dir***config\vob\vob_scrubber_params** (Windows) to the VOB storage directory of the replica. This creates a parameter file specific to the VOB.

b. Make this new file writable.

c. Edit the oplog line in this file. For example, to keep oplog entries for two months (62 days):

```
oplog -keep 62
```

CAUTION: If a replica's oplog entries are scrubbed before they are included in an update packet, you cannot export update packets from the replica. This is a serious error and compromises the integrity of the entire VOB family.

export_sync Scrubbing

export_sync records are not necessary for normal synchronization operation. They are different from export event records, which also record synchronization exports and are included in output from the **lshistory** command and the History Browser.

export_sync records are date-based records used by the **recoverpacket** command to reset a replica's epoch number matrix. If you do not use this packet recovery method (because you use **chepoch –actual** or **lsepoch/chepoch**), you can scrub these records aggressively. If you use the **recoverpacket** command, you must keep export_sync records for the number of days that elapse between VOB backups. (See *Recovering from Lost Packets* on page 182.)

By default, the **vob_scrubber_params** file has no entry for export_sync records, and these records are scrubbed with the same frequency as oplog entries. If you want to scrub export_sync records at a different frequency than oplog entries, you can set the export_sync parameter in the **vob_scrubber_params** file. For more information, see the **vob_scrubber** reference page.

2.5 Responsibilities of MultiSite Administrators

A MultiSite administrator must do the following:

- Help determine and implement the ClearCase and MultiSite use models

  When a new project is set up, the administrator works with project managers to determine which replicas master various objects. The administrator also changes mastership when necessary, schedules merges, copies triggers from replica to replica, and monitors label creation.

- Monitor MultiSite synchronization and replica creation

  Administrators must check the storage bays to make sure that packets are not accumulating. On UNIX, include the administrator's e-mail address in the **ADMINISTRATOR** entry in the **shipping.conf** file. On Windows, include the administrator's e-mail address in the **MultiSite Control Panel**.

  It is important to prevent two or more replicas of the same VOB from being mounted on the same host—one host can belong to only one region and each region can contain only one replica. Accordingly, do not assign *public* VOB-tags in the same ClearCase *registry region* to multiple replicas of the same VOB.

  See *VOB Objects and Replica Objects* on page 23 for information about how VOBs and VOB replicas are listed in the ClearCase storage registry and Chapter 12, *Using MultiSite for Interoperability* for information about using multiple replicas at one site.

- Monitor ClearCase and system log files

  Error and status messages are written to the **shipping_server_log** file and (on Windows) the Event Viewer. For more information about error logs, see *Troubleshooting Tips* on page 164.

- Install new versions of ClearCase and MultiSite and new patches

  Patches and information about new versions are available on the Rational Software Web site. Install the Mandatory and Recommended patches for your architecture.

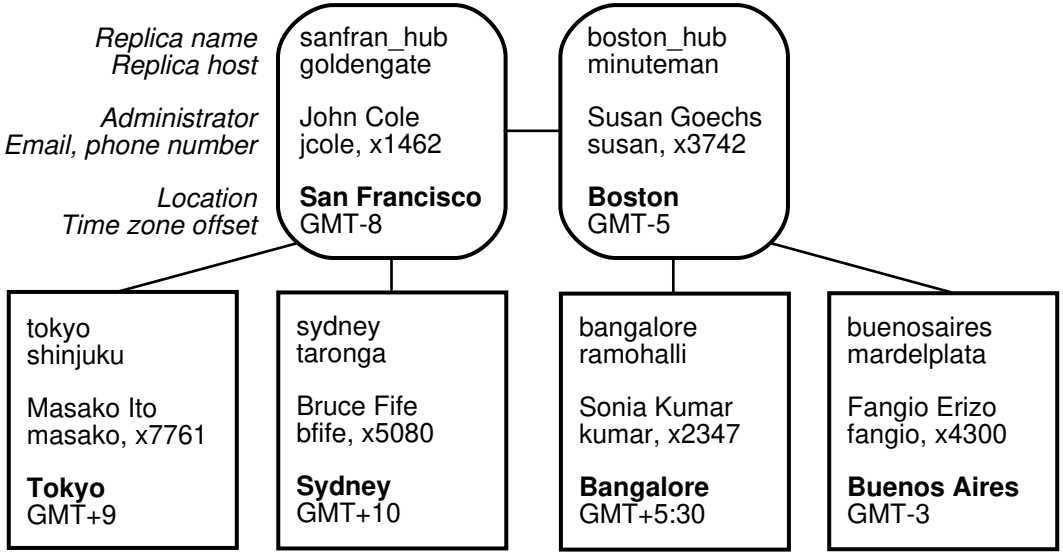
  Compatibility issues for versions of ClearCase and MultiSite are described in the *Release Notes* for Rational ClearCase and ClearCase MultiSite.

- Coordinate issues with all other MultiSite administrators responsible for replicas in the VOB family

  After initial setup and synchronization of replicas, administrators also must coordinate recovery efforts, which may involve exchanges of update packets, and changes of mastership, which require the administrator at the master replica to transfer mastership to the replica that needs to master the objects.

  We recommend that you create a representation of your MultiSite deployment. For example, Figure 17 shows information and the synchronization pattern for a VOB family.

Figure 17 VOB Family Information



➤ Ensure that VOB replicas receive any necessary special handling

Restoring a VOB replica's storage directory from backup is a significant event in the life of a VOB family. Failure to follow the procedure described in the section *Restoring a Replica from Backup* on page 191 leads to irreparable inconsistencies among the VOB's replicas.

There are no special requirements for backing up a VOB replica's storage directory. Use the instructions in the *Administrator's Guide* for Rational ClearCase for backing up a VOB.

Other ClearCase administrative procedures require special steps for replicated VOBs. The procedures in the *Administrator's Guide* for Rational ClearCase describe these steps.

3 MultiSite Command Set

This chapter summarizes MultiSite commands and the ClearCase commands that display MultiSite information. Reference pages for the MultiSite commands are available in Chapter 13, *MultiSite Reference Pages*, and are also available online:

- On UNIX, the MultiSite **multitool man** command displays MultiSite reference pages in either HyperHelp or ASCII format.
- On Windows, the MultiSite **multitool man** command displays reference pages in Windows Help.
- On both platforms, the MultiSite Help file includes the MultiSite reference pages in this manual.

3.1 Location of MultiSite Programs

The MultiSite installation places programs and configuration files in the ClearCase installation area on a host. (*ccase-home-dir* refers to both the ClearCase and MultiSite installation directory).

On UNIX, MultiSite programs are located in the *ccase-home-dir*/**bin**, *ccase-home-dir*/**etc**, and *ccase-home-dir*/**config/scheduler/tasks** directories. On Windows, MultiSite programs are located in *ccase-home-dir***bin** and *ccase-home-dir***config\scheduler\tasks**.

3.2 multitool Use

The **multitool** program is very similar to the ClearCase **cleartool** program:

- It has a set of subcommands that perform product functions, such as replica creation, synchronization, and management; *mastership* of objects stored in VOB databases; and failure recovery.

  Some **multitool** subcommands are also available in **cleartool**.

- Command options can always be abbreviated to three characters and sometimes fewer, as indicated in the reference pages.

- You can use **multitool** in *single-command mode*. For example:

  multitool rename replica:original boston_hub

  Also in *interactive mode*:

  multitool
  `multitool>` **rename replica:original boston_hub**
  `multitool>` **quit**

- It has online help facilities. The **help** command displays syntax summaries, and the **man** command displays reference pages:

  multitool help chreplica
  ```
  Usage: chreplica [-c comment | -cfile pname | -cq | -cqe | -nc]
            [-host hostname]
            [-preserve | -npreserve]
            [-isconnected | -nconnected] replica-selector
  ```

  multitool man chreplica
  ...on Windows, Windows Help displays the reference page

  ```
  chreplica
  ==========
  Changes the properties of a replica

  APPLICABILITY
  ...
  ```

multitool Subcommands

The following sections describe the different kinds of **multitool** subcommands. The tables in each section show whether the command has a **cleartool** equivalent and whether a view context is required when you invoke the command.

Commands Copied from ClearCase

These commands were copied from **cleartool** and are documented only in the *Command Reference*, except for **apropos**, which is also documented in this manual.

Table 5 multitool Subcommands Copied from ClearCase

Command	cleartool equivalent	View context required?	Description
apropos (UNIX)	Yes	No	Displays **multitool** command information
cd	Yes	No	Changes current working directory
describe	Yes	Yes (file-system objects)	Describes a replica's *VOB database* object
help	Yes	No	Displays **multitool** command syntax
man	Yes	No	Displays a MultiSite reference page
pwd	Yes	No	Prints working directory
quit	Yes	No	Ends interactive **multitool** session
rename	Yes	No	Renames a *replica*
shell	Yes	No	Creates subprocess to run shell or program

Replica Creation, Synchronization, and Management

multitool includes commands that set up new replicas of VOBs, change their characteristics, and change their contents by importing update packets.

Table 6 Replica Creation, Synchronization, and Management Commands

Command	cleartool equivalent	View context required?	Description
chreplica	No	No	Changes the properties of a *replica*
lspacket	No	No	Lists one or more *packet* files created by **mkreplica** or **syncreplica**
lsreplica	Yes	No	Lists one or more of a VOB's *replicas*
mkreplica	No	No	Creates a new *VOB replica*
rename	Yes	No	Renames a *replica* (command documented in the *Command Reference*)
rmreplica	No	No	Removes a replica
syncreplica	No	No	Synchronizes the current replica with one or more other replicas in its VOB family

Object Mastership

To prevent conflicting changes from occurring at different replicas of a VOB, certain VOB-database objects are assigned a *master replica* (*master*). The initial master of an object is the replica where the object is created. For more information on mastership, see *Enabling Independent VOB Development: Mastership* on page 7.

Table 7 Object Mastership Commands

Command	cleartool equivalent	View context required?	Description
chmaster	Yes	Yes (file-system objects)	Transfers *mastership* of a ClearCase object
lsmaster	Yes	Yes	Lists objects mastered by a replica
reqmaster	Yes	Yes	Requests mastership or set access controls for mastership requests

Failure Recovery

Each replica of a VOB uses an *epoch number matrix* to track its own state *and* the state of all other replicas. (Because replicas are always changing, a replica knows what changes have been made to itself; but it can have only an estimate of the states of other replicas.) Each time a replica sends an update packet, it updates its own epoch number matrix, under the assumption that the packet will be delivered to its destinations and applied to the appropriate replicas. For more information, see *VOB Operations and the Oplog* on page 24.

multitool includes the following failure-recovery commands, for use when this assumption of successful delivery does not hold true:

Table 8 Failure-Recovery Commands

Command	cleartool equivalent	View context required?	Description
chepoch	No	No	Changes a replica's epoch number matrix
lsepoch	No	No	Lists a replica's epoch number matrix
recoverpacket	No	No	Resets epoch number matrix so lost packets are resent (required when a packet is lost or unusable)
restorereplica	No	No	Restores *VOB replica* from backup. This command places a replica in a special state, in which it sends epoch number matrix corrections to other replicas. The replica cannot be used for normal development work until it receives special updates that inform it of the current states of other replicas.

3.3 View Contexts and VOB Mounts

The principal MultiSite commands do not require a view context or mounting of the VOB replicas being processed. This facilitates use by administrators and automation of MultiSite operations through the **schedule** command.

There are some advantages to running MultiSite commands in a view, with the VOB mounted:

- Simpler command syntax. If your current working directory is within a VOB, many commands process that VOB, eliminating the need to use the *@vob-selector* suffix in command arguments.
- Better diagnostics. If a **syncreplica –import** command fails when running in a view, it produces diagnostics that include pathnames, which makes troubleshooting easier.

3.4 Specifying VOBs and Replicas in Commands

ClearCase commands use the **vob:** prefix to operate on the current VOB replica. ClearCase and MultiSite commands use the *@vob-selector* suffix to specify the replica that is mounted at a particular VOB-tag. This suffix indicates which replica's database is to be used by the command.

The **multitool mkreplica** command uses the **–vreplica** option to specify a particular replica within a VOB family.

3.5 Additional MultiSite Commands

The MultiSite commands that are not built in to **multitool** are listed in Table 9.

Table 9 Additional MultiSite Commands

Command	Location under *ccase-home-dir*	Description
epoch_watchdog	**config/scheduler/tasks** (UNIX) **config\scheduler\tasks** (Windows)	Checks whether a replica's epoch numbers have rolled back when the replica is not in restoration mode; for use in **schedule** commands.
mkorder	**etc** (UNIX) **bin** (Windows)	Creates shipping order for use by *store-and-forward*.
notify	**bin**	Mail program for *store-and-forward*.
shipping_server	**etc** (UNIX) **bin** (Windows)	*Store-and-forward* packet transport server.
sync_export_list	**config/scheduler/tasks** (UNIX) **config\scheduler\tasks** (Windows)	Replica-update script using *store-and-forward*; for use in **schedule** commands.
sync_receive	**config/scheduler/tasks** (UNIX) **config\scheduler\tasks** (Windows)	Replica-update script using *store-and-forward*; for use in **schedule** commands and as the receipt handler.

3.6 ClearCase Commands Related to MultiSite

The ClearCase commands in Table 10 manage or display MultiSite information.

Table 10 ClearCase Commands Related to MultiSite

Command	Description
checkout –unreserved –nmaster	Performs a nonmastered checkout, which is an unreserved checkout on a branch not mastered by your current replica.
lscheckout –areplicas	Lists checked-out versions across all replicas of a VOB (Default: lists your current replica's checkouts).
mkattype –shared **mkhltype –shared** **mklbtype –shared**	Creates a *shared type object*.
mkelem –master	Assigns mastership of the **main** branch of the element to the replica in which you create the element. Also, if your config spec contains **mkbranch** rules and you do not specify the **–nco** option with **mkelem**, **mkelem** assigns mastership of these branches to the replica in which you create the element.
vob_scrubber	Scrubs *oplog entries* and export_sync records.

In general, all ClearCase commands obey MultiSite *mastership* restrictions in a replicated VOB. In addition, the following commands work differently in replicated VOBs:

describe
: Lists the master replica of an object. For replicas, branch types, and branches, lists the mastership request setting.

describe vob:*pname-in-vob*
: Lists the replica name and the VOB family feature level.

ln
mkelem
rmname

To change a directory, you must work in the master replica of the branch on which the directory is checked out. Changes to directories include

- Creating a VOB hard link or VOB symbolic link (**ln**)
- Creating a new element (**mkelem**)
- Removing a reference to an element or VOB symbolic link (**rmname**)

mk****type –replace**

If a type object is shared, you cannot change its instance restrictions. For example, you cannot replace a one-per-element branch type with a one-per-branch branch type.

mkeltype –replace

You cannot change the definition of an element type in a replicated VOB.

rmtype eltype:*type-name*

You cannot delete an element type in a replicated VOB.

Using MultiSite

Creating Replicas

4

This chapter describes how to plan and create VOB replicas. Before creating a replica, you must make decisions about branching, mastership, ownership preservation, and method of packet delivery. Be sure to read *ClearCase Use Model* on page 35 and *MultiSite Use Model* on page 38.

4.1 Overview of Replica Creation

You use this three-phase procedure to create new VOB replicas:

1. **Export phase**—At one site, enter a **mkreplica –export** command, which creates a new replica object and a replica-creation packet.
2. **Transport phase**—Send the packet to one or more other sites.
3. **Import phase**—At the other sites, each administrator enters a **mkreplica –import** command, which creates a new VOB replica.

The procedure is similar for different methods of packet delivery and for different platforms. The example in this chapter assumes a high-speed connection between sites, and all replicas are located on UNIX machines. The procedure is the same if all replicas are located on Windows machines or if one replica is on a Windows machine; only the VOB-tags and pathnames are different.

If some replicas in your VOB family will be located on UNIX machines and others will be on Windows machines, be sure to read *Replicating a VOB Between UNIX and Windows* on page 77.

4.2 Timing of Replica Creation

During the export phase of replica creation, the replica creation command locks the VOB and dumps the VOB database. The VOB is locked for the entire length of time the command runs. While the VOB is locked, read-only operations can occur in the VOB, but write operations cannot. (For example, these operations fail: checkins and checkouts, **chepoch –actual** commands, label creation, builds, imports of update packets, VOB snapshots, and scheduled backups.)

Therefore, you need to schedule the export phase of replica creation during nonbusiness hours for your site. You must also cancel any scheduled exports, imports, VOB snapshots, and backups for the duration of the export phase.

4.3 Notes on Different Transport Methods

If your sites have a high-speed connection, you can take advantage of the MultiSite store-and-forward facility when you create a new replica. If your current site does not have IP connectivity to the site of the new replica, you can use a file-based packet transfer method (for example, ftp or email).

Store-and-Forward Method

The following sections describe issues you must consider when you use the store-and-forward method.

Communication Between Replica Hosts

The hosts must be able to communicate with each other. If your network uses host names, the sending host must be able to resolve the receiving host's name to an IP address. To accomplish this, you may have to update the **hosts** file, **hosts** NIS map, or Domain Name Service. Verify TCP/IP access by using **rcp** on each host to access the other hosts.

NOTE: If hosts in your network are known only by their IP addresses, you can use the IP addresses instead of host names, and no resolution is necessary.

Limiting the Size of a Packet

The **mkreplica** command fails if it tries to create a packet larger than the size supported by your system. To prevent this problem and improve reliability, use the **–maxsize** option to divide the replica-creation packet into multiple packets:

multitool mkreplica –export –maxsize 1g ...

For information about default packet size limits, see the **mkreplica** reference page.

Transport Options

When you enter the **mkreplica –export** command, you can use either the **–fship** option to send the packet immediately, or the **–ship** option to store the packet in the outgoing shipping bay. With **–ship**, you must invoke the **shipping_server** to send the packet.

The outgoing packet is stored in the **outgoing** subdirectory of a storage bay. By default, **mkreplica** uses the default storage bay (*ccase-home-dir***/shipping/ms_ship** on UNIX and *ccase-home-dir***\var\shipping\ms_ship** on Windows).

The **incoming** and **outgoing** subdirectories of storage bays contain packets waiting for transport or processing. All shipping operations look for packets in these subdirectories. At the receiving site, the incoming packet is stored in the **incoming** subdirectory of a storage bay.

Notes on Using Tape or a File-Based Transfer Method

When you use the **–tape** option (UNIX) or a file-based method for transport, you may need to use the **–maxsize** option to prevent the tape from filling up or to make sure the file is a manageable size. In this example, the administrator writes the replica-creation packet to tape, using the **–maxsize** option. The **mkreplica** command prompts for additional tapes if necessary.

```
MINUTEMAN% multitool mkreplica –export –work /usr/tmp/wk  –tape /dev/tape \
–maxsize 75m goldengate:sanfran_hub@/vobs/dev
Enabling replication in VOB.
Comments for "sanfran_hub":
First time replication for dev VOB; Creating new replica, sanfran_hub, on host goldengate
.

Please insert a tape to hold packet number 1.
When ready, enter 'proceed' (proceed/abort) [proceed] <RETURN>
Generating packet number 1...
Dumping database...
  . . .
Dumper done.
```

4.4 Replica-Creation Scenario

The replica-creation example in this section uses a fictional company whose software development takes place in Boston, Massachusetts and in a new development office in San Francisco, California. Work is about to begin on a new release.

Planning the Rules of the Road

The organization uses a common ClearCase software development strategy:

- Individual subprojects, and often individual developers, use separate subbranches. The *auto-make-branch* facility is used in all config specs, to place changes on the appropriate branches. For example:

  ```
  element * CHECKEDOUT
  element * .../sanfran_main/LATEST
  element * SANFRAN_BASE -mkbranch sanfran_main
  element * V1.0 -mkbranch sanfran_main
  element * /main/0 -mkbranch sanfran_main
  ```

- The **v2.0_integration** branch is reserved for integration of the work done at the various sites. To prepare for an internal baseline or an external release, the project manager merges selected development subbranches into the **v2.0_integration** branch.

- When necessary, developers merge changes from the **v2.0_integration** branch to their subbranches, to bring themselves up to date with changes occurring on the integration branch.

With Rational ClearCase MultiSite, the organization can continue to use this strategy. The Boston replica masters the **v2.0_integration** branch. The San Francisco replica masters a branch named **sanfran_main**, as well as any additional subbranches of **sanfran_main** that may be needed to organize the work there. The project manager at the Boston site merges changes from the **sanfran_main** and **boston_main** branches into the **v2.0_integration** branch, so that the release engineers can build the product using the latest changes.

Relevant characteristics of the two replicas:

Boston replica

Host name: **minuteman** (UNIX)
Replica name: **boston_hub**
VOB storage directory: **/vobstg/dev.vbs**
VOB-tag: **/vobs/dev**
Config spec for development:

```
element * CHECKEDOUT
element * .../boston_main/LATEST
element * BOSTON_BASE -mkbranch boston_main
element * V1.0 -mkbranch boston_main
element * /main/0 -mkbranch boston_main
```

Config spec for integration:

```
element * CHECKEDOUT
element * .../v2.0_integration/LATEST
element * BOSTON_BASE -mkbranch v2.0_integration
element * V1.0 -mkbranch v2.0_integration
element * /main/0 -mkbranch v2.0_integration
```

San Francisco replica

Host name: **goldengate** (UNIX)
Replica name: **sanfran_hub**
VOB storage directory: **/vobstg/dev.vbs**
VOB-tag: **/vobs/dev**
Config spec for development:

```
element * CHECKEDOUT
element * .../sanfran_main/LATEST
element * SANFRAN_BASE -mkbranch sanfran_main
element * V1.0 -mkbranch sanfran_main
element * /main/0 -mkbranch sanfran_main
```

The company has not yet merged its user/group databases, so the two replicas cannot be ownership-preserving. There is IP connectivity between the two offices, so the Boston administrator can use the MultiSite store-and-forward facility to create the new replica.

Prerequisites

Before you create a new replica, you must perform these steps at the original site:

1. Make sure MultiSite licenses are installed.

   After you enter the **mkreplica –export** command, developers at the original site cannot access the VOB without a MultiSite license (in addition to a ClearCase license).

   clearlicense –product MultiSite

   ```
   Licensing information for MultiSite.
   License server on host "cclicense".
   Running since Thursday 07/01/00 12:27:28.

   LICENSES:
   Max-Users  Expires       Password [status]
   300          none      34ms5678.901234c5.67 [Valid]
   ...
   ```

2. Apply a version label, from which development work at the new replica will branch.

   In the standard ClearCase manner, a consistent set of source versions (a baseline) is identified by a version label. The VOB administrator creates label type **SANFRAN_BASE** and attaches it to all the **/main/LATEST** versions in the original VOB. The changes at **sanfran_hub** are made on **sanfran_main** branches; all these branches are created at **SANFRAN_BASE** versions.

3. Rename the original replica appropriately.

   Even though the original VOB has not yet been replicated, its VOB database still has a *VOB replica object*, named **original**:

   ```
   MINUTEMAN% cleartool lsreplica –invob /vobs/dev
   For VOB replica "/vobs/dev":
   15-Aug.14:19    susan         replica "original"
   ```

The administrator renames the VOB replica object to **boston_hub**:

```
MINUTEMAN% multitool rename replica:original boston_hub
Renamed replica from "original" to "boston_hub".

MINUTEMAN% cleartool lsreplica –invob /vobs/dev
For VOB replica "/vobs/dev":
15-Aug.14:19    susan         replica "boston_hub"
```

4. Make sure the VOB is not locked.

   Step #6 locks the VOB; an error occurs if the VOB is already locked.

   ```
   MINUTEMAN% cleartool lslock vob:/vobs/dev
   MINUTEMAN%    (null output indicates VOB is not locked)
   ```

5. Determine the size of the VOB database and source pools.

   The directory you specify with the **–workdir** option must be on a partition that has enough free space to hold the VOB database and the VOB source pools. You must have write permission on its parent directory, and the directory you specify must not exist.

   To determine the size of the VOB database and source pools, use **cleartool space**:

   ```
   cleartool space /vobs/dev
   Use(Mb)  %Use  Directory
   ...
   1429.0    17%  VOB database /vobstg/dev.vbs/db
   ...
   189.5      2%  source pool /vobstg/dev.vbs/s/sdft
   ...
   ```

   In this example, the work directory must have at least 1.62 GB of free space.

Export Phase

These steps take place at the original site.

6. Enter the export form of the replica-creation command. See the **mkreplica** reference page for information about restrictions on the command.

   In this example, the administrator uses the **–fship** option to send the packet immediately.

```
MINUTEMAN% multitool mkreplica –export –work /tmp/ms_wkdir \
–fship goldengate:sanfran_hub@/vobs/dev
Enabling replication in VOB.
Comments for "sanfran_hub":
First time replication for dev VOB
Creating new replica, sanfran_hub, on host goldengate
.
Generating replica creation packet
/usr/atria/shipping/ms_ship/outgoing/repl_boston_hub_16-Aug-00.09.49.36_14
075_1
- shipping order file is
/var/adm/atria/shipping/ms_ship/outgoing/sh_o_repl_boston_hub_16-Aug-00.09
.49.36_14075_1
Dumping database...
...
Dumper done.
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/repl_boston_hub_16-Aug-00.09.49.36_14
075_1
```

7. Back up the original VOB.

   This backup records the fact that the VOB is replicated. If you have to restore a VOB replica from a backup copy that was made before the VOB was replicated, the MultiSite replica restoration procedure fails. (Although the **restorereplica** command may succeed, you will not be able to import update packets from other replicas because the original VOB is marked as unreplicated.)

8. (optional) Verify the replica-related changes.

   These commands show the work you've done so far. The **mkreplica** command creates a new *replica object* in the VOB database. This object is similar to the *VOB object* that represents the entire VOB in the database, and its properties are listed by the **lsreplica** command.

```
MINUTEMAN% multitool lsreplica –invob /vobs/dev
For VOB replica "/vobs/dev":
15-Aug.14:19   susan       replica "boston_hub"
16-Aug.09:49   susan       replica "sanfran_hub"
```

   MultiSite commands process replica objects similarly to the way that ClearCase commands process *type* objects. The **rename** command renames a replica object, as described in Step #3. The **cleartool lshistory** lists events associated with replica objects.

```
MINUTEMAN% cleartool lshistory replica:boston_hub@/vobs/dev
16-Aug.09:45   susan     rename replica "boston_hub"
"Changed name of replica from "boston" to "boston_hub"."
15-Aug.14:24   susan     rename replica "boston_hub"
"Changed name of replica from "original" to "boston"."
15-Aug.14:19   susan     make attribute "FeatureLevel" on replica
"boston_hub"
"Added attribute "FeatureLevel" with value 2."
15-Aug.14:19   susan     create replica "boston_hub"
```

CAUTION: Do not modify any properties of the new replica at this point. If you must change any properties, you must first import the replica-creation packet at the new site, export an update packet from the new replica, and import the packet at the original site.

Transport Phase

9. Send the replica-creation packet to the new site. This process differs depending on the options you used in Step #6:

- If you used **–fship** in Step #6, the packet was sent to the new site immediately.
- If you used **–ship**, you must run **shipping_server** to send the packet to the new site. For example:

  ```
  MINUTEMAN% /usr/atria/etc/shipping_server –poll
  ```

- If you used **–tape** or wrote the packet to a file, you must transport the tape or file to the new site.

Import Phase

Complete these steps at the new replica's site.

10. Verify the packet's arrival by entering the **lspacket** command on the receiving host.

By default, **lspacket** searches all the MultiSite storage bays for packets. For example, if host **goldengate** is the receiving host:

```
GOLDENGATE% multitool lspacket
Packet is:
/usr/atria/shipping/ms_ship/incoming/repl_boston_hub_16-Aug-00.09.49.36_14
075_1
Packet type: Replica Creation
VOB family identifier is: 12a3456b.78c901d2.e3ab.45:67:89:c0:1d:e2
Comment supplied at packet creation is:
Packet intended for the following targets:
sanfran_hub
The packet sequence number is 1
```

11. Enter the import form of the replica-creation command.

Notes on replica creation:

- This replica is not ownership-preserving, so the user who executes this command becomes the owner of the new VOB replica and all elements in it. This user's primary group is the group for all elements. Typically, administration is easier if this user is not the **root** user or a member of the ClearCase administrators group.
- As described in Step #5, the work directory must have at least 1.62 GB of free space.
- You can bypass the prompt step during replica creation by specifying the **–vreplica** option with the new replica's name. This example does not specify that option.
- You must specify the pathname of the incoming packet as listed by the **lspacket** command.

```
GOLDENGATE% multitool mkreplica –import –npreserve –work /tmp/wk \
–tag /vobs/dev  –public –password 1234xyz –vob /vobstg/dev.vbs \
/usr/atria/shipping/ms_ship/incoming/repl_boston_hub_16-Aug-00.09.49.36_14075_1
The packet can only be used to create replica "sanfran_hub"
- VOB family is 87f6265b.72d911d4.a5cd.00:01:80:c0:4b:e7
- replica OID is 0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7
Should I create this replica? [no] yes
Comments for "sanfran_hub":
replica of /vobs/dev from Boston
.
Processing packet
/usr/atria/shipping/ms_ship/incoming/repl_boston_hub_16-Aug-00.09.49.36_14
075_1...
Loading database...
Dumped schema version is 53
55 events loaded.
77 pass 2 actions performed.
Loader done.
Registering VOB mount tag "/vobs/dev"...
VOB replica successfully created.
Host-local path: goldengate:/vobstg/dev.vbs
Global path:     /net/goldengate/vobstg/dev.vbs
VOB ownership:
owner purpledoc.com/jcole
group purpledoc.com/user
```

12. Delete the replica-creation packet. (Update packets are deleted automatically.)

13. (Only if new replica is ownership-preserving) Send an update packet to all other replicas in the VOB family.

    If you create an ownership-preserving replica, inform other replicas in the VOB family of this property. For example, if **sanfran_hub** were ownership-preserving, you would enter this command:

    ```
    GOLDENGATE% multitool syncreplica –export –c "ownership-preserving"  –fship
    boston_hub
    ...
    ```

14. Make the new replica self-mastering. See *Transferring Mastership of a Replica Object* on page 128 for the procedure.

    You must complete this step before you can set the new replica's feature level or enable requests for branch mastership at the replica.

15. Set the feature level of the new replica to the feature level of the version of Rational ClearCase running on the replica host.

    To determine the feature level of the version of ClearCase, enter the **cleartool –ver** command on the replica host to display the ClearCase version. Then consult the *Release Notes* for the feature level associated with the version.

    To set the new replica's feature level, enter a **chflevel** command on the replica host:

    cleartool chflevel –replica *feature-level replica-selector*

    For example:

    ```
    cleartool chflevel –replica 2 sanfran_hub@/vobs/dev
    Replica feature level raised to 2.
    ```

16. Send an update packet to all other replicas in the VOB family, to inform them of the new replica's feature level. For example:

    ```
    GOLDENGATE% multitool syncreplica –export –c "new feature level" –fship boston_hub
    ...
    ```

17. Create a branch type for work in the new replica.

    The San Francisco developers work on the **sanfran_main** branch type.

    ```
    GOLDENGATE% cleartool mkbrtype sanfran_main
    Comments for "sanfran_main":
    sanfran branch for work on dev project
    .
    Created branch type "sanfran_main".
    ```

    Subbranches named **sanfran_main** are created as necessary. The following config spec automates the creation of the **sanfran_main** branches:

    ```
    element * CHECKEDOUT
    element * .../sanfran_main/LATEST
    element * SANFRAN_BASE -mkbranch sanfran_main
    element * V1.0 -mkbranch sanfran_main
    element * /main/0 -mkbranch sanfran_main
    ```

    This config spec is defined in terms of a branch type (**sanfran_main**) that is mastered by replica **sanfran_hub**, and label type (**SANFRAN_BASE** and **V1.0**) that are mastered by replica **boston_hub**. The San Francisco developers cannot make any changes in the **SANFRAN_BASE** labels, but there is no reason for them to do so.

18. Verify the mastership of the new branch type.

```
GOLDENGATE% cleartool describe brtype:sanfran_main@/vobs/dev
branch type "sanfran_main"
  created 16-Aug-00.18:12:23 by John Cole (jcole.user@goldengate)
  "sanfran branch for work on dev project"
  master replica: sanfran_hub@/vobs/dev
...
```

19. (For IP-connected replicas) At each site, mark any sibling replicas that have IP connectivity to the current replica. For more information, see *Setting the Connectivity Property* on page 116.

    At the **boston_hub** replica:

    multitool chreplica –isconnected sanfran_hub@/vobs/dev
    ```
    Updated replica information for "sanfran_hub".
    ```

    At the **sanfran_hub** replica:

    multitool chreplica –isconnected boston_hub@/vobs/dev
    ```
    Updated replica information for "boston_hub".
    ```

20. Begin development.

    Developers in San Francisco can access the new replica in the same way they would access an unreplicated VOB.

4.5 Replicating a VOB Between UNIX and Windows

This section describes issues involved in setting up UNIX and Windows replicas at different sites. If you plan to use MultiSite at a single location for interoperability between UNIX and Windows, see Chapter 12, *Using MultiSite for Interoperability*.

If your sites do not have an IP connection, you must use electronic mail, CD-ROMs, tapes, or diskettes to transfer packets. You may have to solve compatibility problems if you choose to use tapes or diskettes. With electronic mail, you can use compatible encoding and compression methods. However, differences between UNIX and Windows VOBs are handled automatically during packet import.

The most important problems you must prevent are file names that differ only in how they are capitalized, and differences in use of line terminators. If case-sensitive file names are used at one

replica and case-insensitive file names are used at another replica, errors can occur during synchronization. Differences in use of line terminators between UNIX and Windows editors cause unexpected behavior during file comparisons and merges. Even if the contents of the files are identical, different line terminators indicate differences in the files and require a merge.

The *Administrator's Guide* for Rational ClearCase describes these problems and their solutions in detail. Be sure to read it before setting up UNIX and Windows replicas.

5 ClearCase Feature Levels

This chapter describes ClearCase *feature levels* and how to raise the feature level of a replica and a VOB family.

5.1 Overview of Feature Levels

Feature levels allow different replica hosts in a VOB family to run different versions of Rational ClearCase. New versions of ClearCase may introduce features that are incompatible with old versions, but administrators may not be able to upgrade all replica hosts at the same time. Feature level control enables you to upgrade replica hosts at different times and to prevent developers from using new ClearCase features that are not meaningful to replicas on hosts running earlier versions.

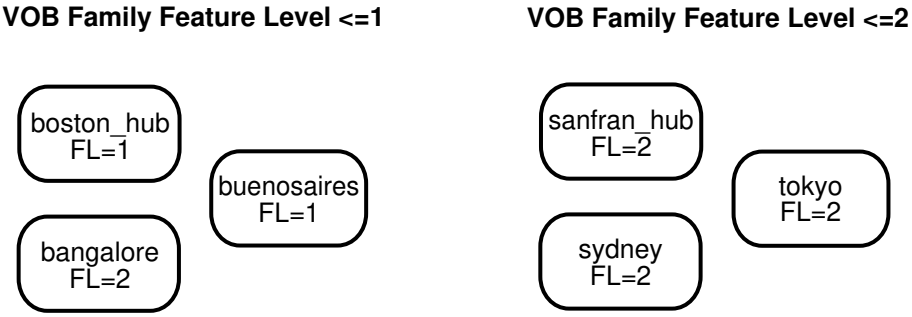
Each version of ClearCase has a feature level. Each VOB family has a single feature level called the family feature level. Each replica in the family has a feature level called the replica feature level. Thus, each VOB family has one family feature level and possibly several replica feature levels. The family feature level determines which ClearCase features can be used by all of the replicas in the family. The following constraints are enforced:

- The replica feature level is less than or equal to the feature level of the version of ClearCase installed on the replica's server host.

  Different replicas on the same server host can have different feature levels.

- The family feature level is less than or equal to the lowest replica feature level found among replicas in the VOB family. Figure 18 shows an example.

Figure 18 VOB Family Feature Levels



The general procedure for raising a family's feature level is as follows:

1. Install the new version of ClearCase on the server hosts of replicas in the VOB family.
2. Raise the feature level of each replica in the VOB family. See *Raising the Replica Feature Level*.
3. Raise the feature level of the VOB family. See *Raising the VOB Family Feature Level* on page 82.

You can complete these steps incrementally and over a period of days or weeks, if necessary. Variations are possible; for example, if a VOB family has replicas **R1** and **R2** on servers **S1** and **S2**, respectively, you can install a new version of ClearCase on **S1** and raise **R1**'s replica feature level before installing the new version on **S2**. However, you can complete Step #3 only after you have raised all replicas in the family to the new feature level.

For information about the feature level associated with the current version of ClearCase, and the list of features that are disabled until the VOB family feature level is raised, see the *Release Notes* for Rational ClearCase and ClearCase MultiSite.

5.2 Raising the Replica Feature Level

There are two important rules related to raising a replica's feature level:

1. If the current family feature level is less than or equal to 1, the first replica in a VOB family whose feature level is raised must be the replica that masters the VOB object.
2. The replica must be self-mastering.

To raise the replica feature level:

1. After installing the new version of ClearCase on a server host, determine which replica masters the VOB object:

   cleartool describe vob:*vob-tag*

   If the replica whose feature level you want to raise first does not master the VOB object, transfer mastership, and then export an update packet to the replica whose feature level you want to raise:

   multitool chmaster *replica-name* **vob:***vob-tag*

   multitool syncreplica –export –fship *replica-name@vob-tag*

   At the receiving replica, import the packet:

   multitool syncreplica –import –receive

2. Determine whether the replica is self-mastering:

   cleartool describe replica:*replica-name@vob-tag*

3. If the replica is not self-mastering, convert it to a self-mastering replica. See *Transferring Mastership of a Replica Object* on page 128.

4. Raise the feature level of the replica. Enter this command on the replica host:

   cleartool chflevel –replica *feature-level* **replica:***replica-name@vob-tag*

5. Export update packets to all other replicas in the VOB family.

6. (optional) Change mastership of the replica back to the original master replica.

5.3 Raising the VOB Family Feature Level

There are two variants of the procedure for raising the family feature level:

- Raising the feature level of a VOB family in which all replicas send update packets (bidirectional synchronization). See *VOB Families with Bidirectional Synchronization* on page 82.
- Raising the feature level of a VOB family in which one or more replicas receive update packets, but do not send them (unidirectional synchronization). See *VOB Families with Unidirectional Synchronization* on page 82.

VOB Families with Bidirectional Synchronization

After raising the feature level of all replicas in the VOB family:

1. Raise the family feature level. Enter this command at the replica that masters the VOB object:

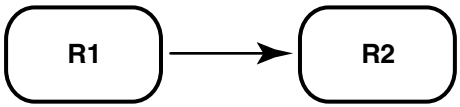
   cleartool chflevel –family *feature-level* **vob:***vob-tag*

2. Export update packets to all replicas in the family.

VOB Families with Unidirectional Synchronization

In some VOB families, one or more replicas may be one-way replicas. These replicas import packets, but they do not export packets to any other replicas in the family, and therefore cannot communicate changes in feature level. Because other replicas in the family do not know the current feature level of the one-way replicas, the **chflevel –family** command fails.

For example, consider the case of two replicas, **R1** and **R2**, that constitute a VOB family. **R1** sends update packets to **R2**, but **R2** does not send update packets to **R1**.



R1 is at replica feature level **2**, and **R2** is at replica feature level **1**. Therefore, the family feature level is **1** and cannot be raised. Now suppose **R2**'s replica feature level is raised to **2**. **R2** cannot communicate the change in feature level to **R1** because it does not export update packets.

Because both replicas are now at feature level **2**, the VOB family feature level can be raised to **2**. However, if the R1 administrator issues the command **chflevel -family 2** *vob-selector*, the change fails because **R1** doesn't know that the replica feature level at **R2** has been raised.

In this case, the R2 administrator must inform the R1 administrator of the change in **R2**'s replica feature level. The R1 administrator then uses a special form of the **chflevel** command to raise the VOB family feature level. The general procedure is as follows:

1. The administrator of a one-way replica notifies other replica administrators in the VOB family of a change in replica feature level at the one-way replica.

2. At the replica that masters the VOB object, the administrator enters the following command:

   cleartool chflevel –force –override –family *feature-level* **vob:***vob-tag*

   CAUTION: This form of the **chflevel** command bypasses the constraint that the family feature level is no higher than the lowest known feature level of the replicas in the VOB family. Use it only when you are certain that all replicas in the VOB family are at the same feature level. If you use this command inappropriately, synchronization will fail.

3. At the replica that masters the VOB object, export update packets to all replicas in the family.

5.4 Displaying Feature Levels

To display the feature level of a replica:

➤ Use the command **cleartool describe replica:***replica-name@vob-tag*. For example:

cleartool describe replica:tokyo@\dev

```
replica "tokyo"
  created 20-Aug-00.13:35:37 by John Cole (jcole@goldengate)
  replica type: unfiltered
  master replica: sanfran_hub@\dev
...
  feature level: 2
...
```

➤ On Windows, open the Properties Browser for the replica.

To display the feature level of a VOB family, use the command **cleartool describe vob:***vob-tag*. For example:

cleartool describe vob:/vobs/dev

```
versioned object base "/vobs/dev"
  created 15-Aug-00.14:19:03 by Susan Goechs (susan.user@minuteman)
  master replica: boston_hub@/vobs/dev
  replica name: boston_hub
  VOB family feature level: 2
...
```

NOTE: Before you set the feature level for a newly created replica, its value is recorded as unknown. For example, if you use the **describe** command to show the properties of a new replica, the output looks like this:

cleartool describe replica:sanfran_hub@/vobs/dev

```
...
  feature level: unknown
```

5.5 Feature Levels Error Message

The following error message is printed when a user attempts to use a feature that is not meaningful to sibling replicas:

```
The feature level of the VOB family is not high enough to permit this
operation.
```

6 Synchronizing Replicas

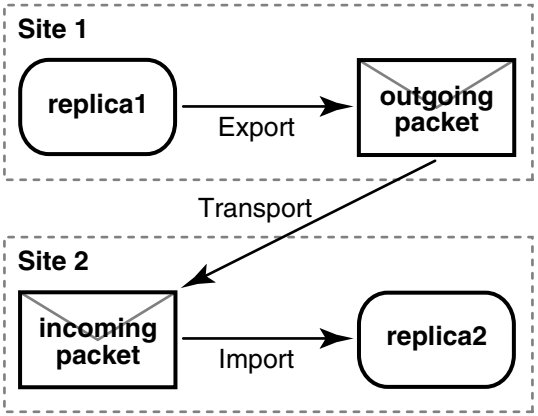
This chapter describes the process of synchronization. Synchronization uses the same export-transport-import procedure that is used during replica creation:

- **Export phase**—At one site, a **syncreplica** (synchronize replica) command is invoked with the **–export** option. This creates a *packet* of data.
- **Transport phase**—The packet is sent to one or more other sites.
- **Import phase**—At the other sites, a **syncreplica** command is invoked with the **–import** option. This applies the changes in the packet to an existing replica.

The **syncreplica** command is optimized for performance; it creates a packet that contains only the information required to update the target replicas specified on the command line.

Figure 19 illustrates the MultiSite replica-synchronization scheme. At Site 1, a **syncreplica –export** command places version data and records of operations from **replica1** into a packet. The packet is sent to Site 2. At Site 2, a **syncreplica –import** command imports the contents of the packet into **replica2**. Note that each synchronization is one-way. If two replicas update each other, two synchronizations are required.

Figure 19 Replica Synchronization



6.1 Synchronization Patterns

Figure 19 shows a simple case, involving one point-to-point update. All updates need not be point to point, however, because they are cumulative. Suppose that the following updates take place among three replicas:

Update 1: Replica 1 sends changes to Replica 2
Update 2: Replica 2 sends changes to Replica 3

There is no need for Replica 1 to update Replica 3 directly, because the changes from Update 1 are included in Update 2. This feature gives administrators flexibility in devising update strategies and patterns. For efficiency, a single update can be targeted at multiple sites, for example, all other replicas in the VOB family.

In general, you can implement any update topology, as dictated by organizational structures, communications/transportation costs, and so on. Figure 20 shows a simple peer-to-peer synchronization update pattern and Figure 21 shows a double-hub hierarchical pattern. See Chapter 2, *Planning a MultiSite Implementation*, for more information.

Figure 20 Peer-to-Peer Synchronization Pattern

Figure 21 Hierarchical Synchronization Pattern

Designing an Update Strategy

Site administrators must design a strategy for sending updates among the various replicas. They must specify an update pattern for the VOB family and an update frequency for each replica. *MultiSite Use Model* on page 38 gives planning guidelines.

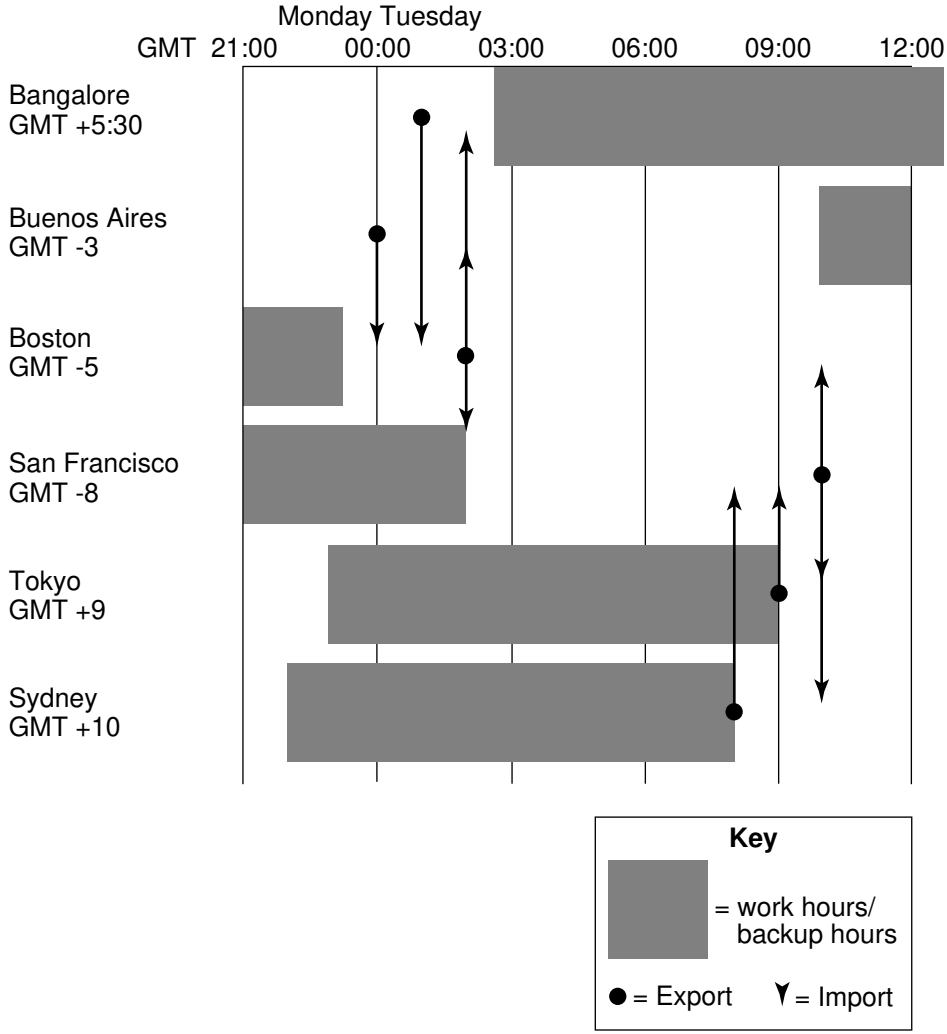
For example, the administrators for the VOB family in Figure 21 make the following decisions:

- The hub replicas, which undergo rapid development, synchronize every hour.
- Each hub replica synchronizes daily with its spoke replicas. Each spoke replica will send an update packet to the hub replica, and then the hub replica will send update packets back to the spoke replicas. Because these packets may be large and take a long time to import, the synchronization must not take place during working hours or during VOB backups.

  NOTE: Synchronization cannot overlap with VOB backup because VOBs must be locked while they are being backed up, and the **syncreplica** command fails if the VOB is locked.
- All sites use receipt handlers to import packets as soon as they are received.

Figure 22 shows the synchronization timeline for the hub-spoke updates (but not the hub-to-hub updates). This timeline accounts for time zone differences and includes extra time to make sure that each synchronization phase completes before another begins.

Figure 22 A Synchronization Export Schedule



6.2 Assumption of Successful Synchronization

The export and import phases of synchronization always occur at different times. A sending replica does not require acknowledgment from a sibling replica that a packet has been received and processed successfully. Instead, the sending replica assumes success. This enables an optimization: subsequent updates from a replica do not include the data sent in previous updates.

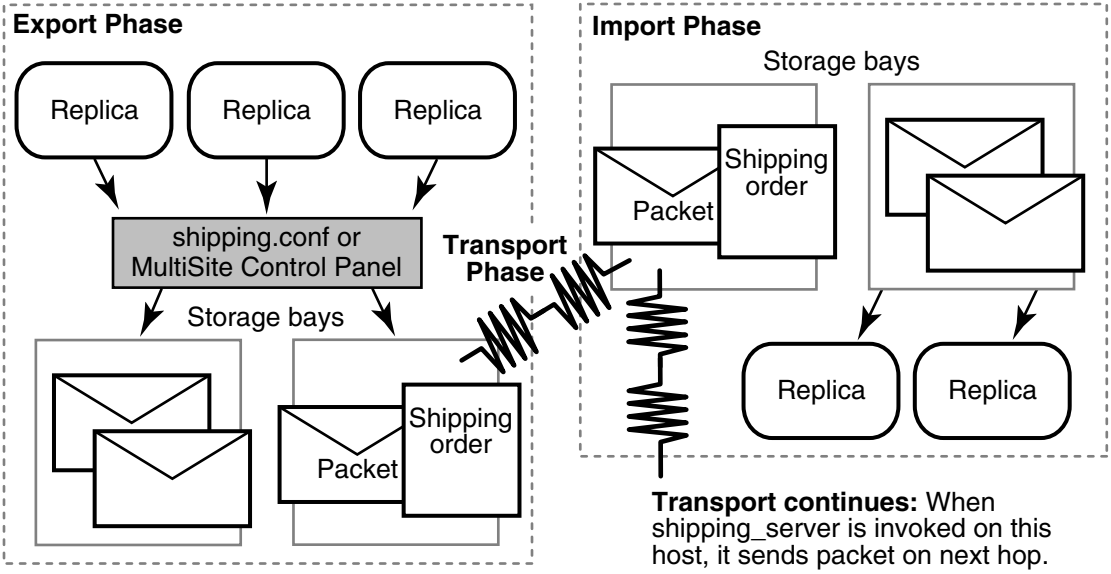
If a failure does occur (for example, a packet is lost in transit or a CD-ROM is unreadable at the sibling replica's site), the sending site must adjust its records to enable the lost data to be resent. For more on this topic, see Chapter 10, *Troubleshooting MultiSite Operations*.

6.3 Transferring Packets with Store-and-Forward

The MultiSite *store-and-forward* facility is a file-transfer service that automates the *transport phase*. This facility can handle packets of any size, can route files through a series of MultiSite hosts, one hop at a time, and includes support for handling data-communications failures. The major components of the store-and-forward facility are illustrated in Figure 23 and described in the following sections.

NOTE: To use store-and-forward, the sending host must be able to communicate with the receiving hosts. To determine whether the hosts can communicate, use the **rcp** command on the sending host to copy a file to the receiving host. If it fails, you may have to update the **hosts** file, **hosts** NIS map, or Domain Name Service before using store-and-forward.

Figure 23 The Store-and-Forward Facility



Packet Storage During the Export and Import Phases

When a physical packet file is exported from a VOB replica and submitted to the *store-and-forward* facility, it is accompanied by a *shipping order* file, which specifies delivery instructions. The packet is stored in one of the *storage bay* directories on the VOB replica host.

Each storage bay directory contains two directories, **incoming** and **outgoing**, which hold the incoming and outgoing packets and their corresponding shipping order files. Shipping operations look in the **incoming** and **outgoing** directories for packets. A default storage bay, **ms_ship**, is created on a host when Rational ClearCase MultiSite is installed there.

NOTE: On Windows, the amount of available space on the disk partition where the shipping bays are located must be at least twice the size of the largest packet that will be stored in the shipping bays. There may be two copies of the same packet in the bay at one time: one on its way to another destination and another waiting to be applied to the replica on the host.

Return bays are similar to storage bays and provide "return-to-sender" storage for packets that could not be delivered successfully. A default return bay, **ms_rtn**, is created on a host when MultiSite is installed there. This bay has two subdirectories, **incoming** and **outgoing**, which hold the incoming and outgoing packets. Shipping operations look in the subdirectories for packets.

Packet Transport

The **shipping_server** program transfers packet files from a storage bay (or return bay) at one site to the corresponding bay at another site.

An explicit command, manual or automated, invokes the **shipping_server** on the sending host. The **shipping_server** process contacts the **albd_server** process on the receiving host, which in turn invokes the **shipping_server** on the receiving host in receive mode. After a TCP/IP connection has been established between the sending and receiving invocations of **shipping_server**, the file is transferred.

Configuring the Store-and-Forward Facility

The settings for the store-and-forward facility are host-specific. You can specify locations of storage and return bays, routing information to support multihop packet delivery, specifications to handle failure-to-deliver situations, receipt handlers, and so on. For more information on specifying settings, see the **shipping.conf** reference page on UNIX or the **MultiSite Control Panel** reference page on Windows.

Submitting Packets to Store-and-Forward

When you generate a replica-creation or update packet, you can specify that the *store-and-forward* facility must deliver it. Both **syncreplica** and **mkreplica** support the following options:

- The **–fship** option places the packet files and shipping order files in one of the host's storage bays, and runs **shipping_server** to send the packet files to their destination host or route them to an intermediate host.
- The **–ship** option places the packet files and shipping order files in a storage bay, but does not invoke **shipping_server**. The packet files are sent the next time the **shipping_server** polls the appropriate bay. For information about setting up **shipping_server** to run automatically, see *Automated Synchronization* on page 104.

Sending Files That Are Not Packets

You can send any file using the store-and-forward facility if you create a shipping order for the file with the **mkorder** utility. You can send the file immediately or wait for the **shipping_server** to send it.

➤ To send a file immediately, use the **–fship** option with **mkorder**:

/usr/atria/etc/mkorder –data /usr/rptgen/brdcst.0702 –fship –copy boston_hub tokyo

➤ To store the file in a shipping bay so that **shipping_server** will send the file the next time it runs, use the **–ship** option:

/usr/atria/etc/mkorder –data /usr/rptgen/brdcst.0702 –ship –copy boston_hub tokyo

NOTE: The shipping order must be located in the same directory as the file.

After you invoke the **mkorder** command, you can delete the original file.

If a file with the same name already exists on the receiving host, the file you send is renamed to *filename*_**1**. If you transmit another file with the same name, it is renamed to *filename*_**2**, and so on.

Setting Up an Indirect Shipping Route

The shipping order for a packet includes the host name of the packet's final destination or several such host names. By default, the store-and-forward facility sends the packet directly to its destination host. You can specify that the packet must be sent to an intermediate host by associating it with a routing hop in the **shipping.conf** file (UNIX) or in the **MultiSite Control Panel** (Windows).

For example:

➤ On a UNIX host, the **shipping.conf** file includes this line:

```
ROUTE  sydney_fw    sanfran_hub boston_hub tokyo
```

➤ On a Windows host, the Routing Information section in the MultiSite Control Panel specifies host **sydney_fw** in the **Next Routing Hop** box and hosts **sanfran_hub**, **boston_hub**, and **tokyo** in the **Destination Hostnames** box.

Any packet whose final destination is host **sanfran_hub**, **boston_hub**, or **tokyo** is forwarded to host **sydney_fw**. At this point, the local host has completed its task, and responsibility for delivering the packet now belongs to **sydney_fw**. Host **sydney_fw** can transmit the packet to its final destination directly, or send it to yet another intermediate host, depending on the settings in its **shipping.conf** file or in the **MultiSite Control Panel**.

NOTE: In a multihop transmission, using the **–fship** option on the original host causes the *first* hop to occur immediately. Subsequent hops occur when **shipping_server** is invoked on the intermediate hosts, which may not be immediately after the packets are received.

Retries, Expirations, and Returned Data

The **shipping_server** makes one attempt to transmit a packet to another host. If the packet cannot be transmitted (for example, because the receiving host is unavailable), **shipping_server** generates an error message and log file entry and exits. Administrators can set up a retry scheme to control its frequency:

- After successful transmission of a packet, **shipping_server** deletes the packet and its shipping order. After a failure, the packet and shipping order remain in the storage bay.
- **shipping_server –poll** transmits all packets it finds in one or more storage bays. Thus, any packets that remain after a transmission failure are sent (if possible) by the next invocation of **shipping_server –poll**.

  The following job definition performs this operation every hour:

```
Job.Begin
   Job.Id: 16
   Job.Name: "Shipping Server Poll"
   Job.Description.Begin:
Every hour, run the shipping server to send out any outstanding orders.
   Job.Description.End:
   Job.Schedule.Daily.Frequency: 1
   Job.Schedule.FirstStartTime: 00:00:00
   Job.Schedule.StartTimeRestartFrequency: 01:00:00
   Job.DeleteWhenCompleted: FALSE
   Job.Task: 13
   Job.Args: -quiet 1 -poll
Job.End
```

  See the **cleartool schedule** reference page in the *Command Reference* and *Automated Synchronization* on page 104.

Attempts to transmit an undelivered packet can continue indefinitely, through repeated invocations of **shipping_server**. However, administrators usually want to fix problems with failed transmissions instead of letting the attempts continue. Accordingly, each shipping order can include an expiration date-time, specified with one of the following:

- The command option **–pexpire**
- (UNIX) An **EXPIRATION** entry in the **shipping.conf** file
- (Windows) A **Packet Expiration** value in the **MultiSite Control Panel** at the sending host

By default, shipping orders expire 14 days after they are created.

When **shipping_server** encounters a shipping order that has expired, it does not attempt to transmit the corresponding packet to its destination. Instead, it does the following:

- It modifies the shipping order to return the packet to the original sending host, where it is placed in a special *return bay.*
- It sends an electronic mail message to one or more addresses on the original sending host. (Another message is sent when the returned packet arrives at the original sending host.)

The return trip may involve multiple hops, as described in *Setting Up an Indirect Shipping Route* on page 93. During such a trip, a packet is placed in the return bay of each intermediate host. Each hop is handled by **shipping_server –poll**, which processes a host's return bay in addition to its storage bays. The expiration time for a packet's return trip is 14 days; a packet that cannot be returned in that interval is deleted.

Error Notification in a Mixed Environment

If a packet is delivered through a Windows host on which e-mail notification is not enabled, a failure on that Windows host means that no notification message is sent by electronic mail. Instead, a message is written to the event log; this message contains a request that the appropriate users be informed of the failure. For information about enabling e-mail notification, see the **MultiSite Control Panel** reference page.

Differentiating Packets with Storage Classes

You can configure the store-and-forward facility to handle updates for different VOBs in different ways. Each packet can be assigned to a *storage class*, and each storage class can have its own *storage bay*, *return bay*, and *expiration period*.

NOTE: On UNIX, a storage class can be assigned several storage bays; in this case, **shipping_server** uses the size of the packet to select one of the bays. Conversely, several storage classes can share one or more storage bays.

You can use multiple storage classes to segregate the packets for VOBs belonging to different groups. By adjusting the operating system permissions on the storage bay directories, you can protect the packets from unauthorized use. You can also use a separate storage class when you use the store-and-forward facility to transfer non-MultiSite files between sites.

For more information on storage classes, see the **shipping.conf** and **MultiSite Control Panel** reference pages.

6.4 Using MultiSite through a Firewall

The MultiSite store-and-forward facility cannot operate through a firewall unless you configure MultiSite differently. Passing through a firewall is usually accomplished by granting access via specific ports and IP addresses. Because store-and-forward picks any available port number on each end to make the connection, there is no single port number (or even small range of port numbers) to which special access can be granted.

This section describes several ways to use MultiSite through a firewall:

- Use an existing electronic mail mechanism as the transport.
- Use the standard **ftp** utility to transport packets.
- Use a custom TCP application.
- (UNIX) Install the store-and-forward software on a host that can communicate through the firewall.

Using Electronic Mail

You can use an existing electronic mail mechanism as the transport. On the sending end, compress and encode the update packet; then send the resulting data to a specific mail alias at the receiving site. On the receiving end, redirect the mail alias to a script that decodes and decompresses the incoming information. To ensure that a mail message is not too large to be delivered, you can generate packets no larger than a specific size by using the **–maxsize** option, the **shipping.conf** file (UNIX), or the **MultiSite Control Panel** (Windows).

Advantages:

- Transport mechanism is well understood and widely available.
- Little effort is required from the system administrator.

Disadvantages:

- No control over routing of data.
- Possibility that messages can be lost without notification.
- Messages can be intercepted easily.
- Less efficient than **ftp** or store-and-forward.

Notes:

- You can write scripts to automate e-mail transport. The sending script creates the update packets, compresses and encodes them, and divides them into multiple small packets so they are not too big for the e-mail process. The script must mark the multiple packets with the correct sequencing. The script then sends the packets to an address at the target replica.

  At the target replica, the account that receives the packets redirects or pipes the packets to a process that reassembles, decodes, and uncompresses the packets, and then places them in the replica's storage bay.

  MultiSite import commands handle out-of-sequence and missing packet problems, so your scripts do not have to address these issues.

- Using **ssh** and **scp** (secure shell and secure copy) provides a secure way to move files through firewalls.

- For security, you must encrypt the packets.

Using FTP

The **ftp** utility can transport packets. On the sending end, the MultiSite administrator or a script creates and compresses the packet, and uses **ftp** to transfer the file to a location outside the firewall. This location, or dropsite, must be accessible by MultiSite administrators at other sites. Receiving sites poll the dropsite, looking for any new files. When new files arrive, the receiving sites retrieve them via **ftp**, decompress them, and process them as usual.

Advantages:

- Transport mechanism is well understood and widely available.
- More reliable and efficient than electronic mail.

Disadvantages:

- Use of a dropsite is required.
- Polling of the dropsite is required.
- More complicated to implement, due to the interactive nature of the **ftp** utility.
- More administration is required because a third system (the dropsite) is used.

Using Custom Software

A custom TCP application can accept data and send it from one site to a waiting application at another site. Guidelines for simple applications that send data are often described in the network programming documentation provided by the vendor. If the sending and receiving applications use a fixed port number, the administrator can configure the firewall to permit access.

Advantages:

- Efficient and reliable.
- No dropsites required.
- Electronic mail-capable network is not required.
- Data interception is more difficult.

Disadvantages:

- Custom coding is required.
- Not as flexible as electronic mail or FTP solutions.

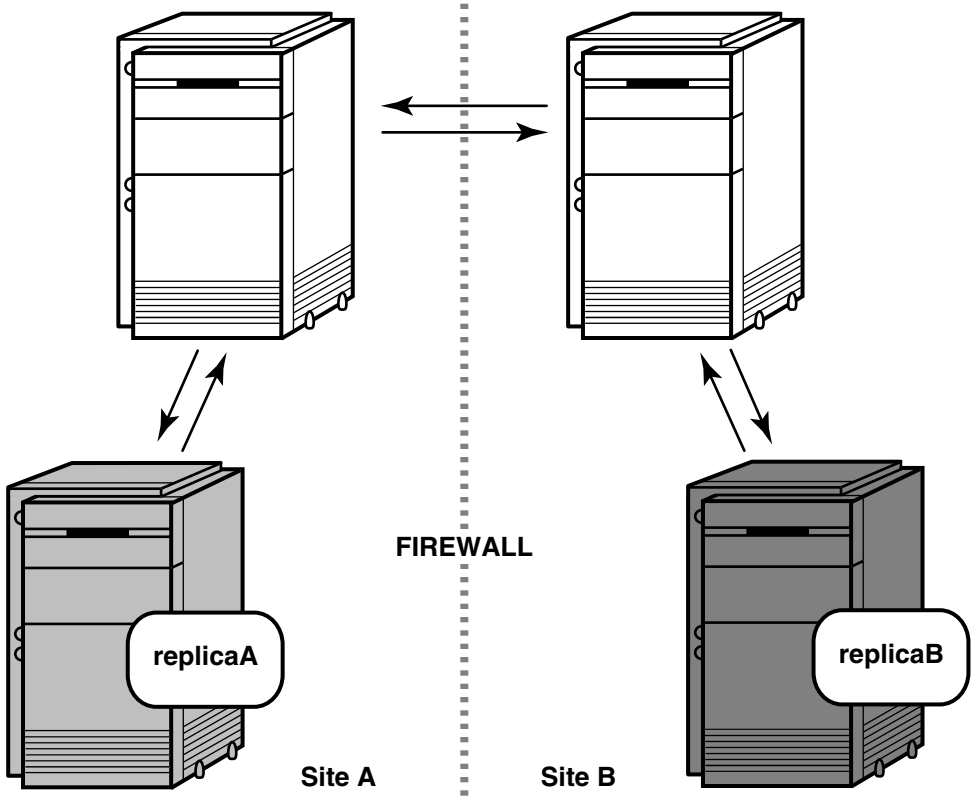
Installing Store-and-Forward on a UNIX Firewall Host

NOTE: Because of security concerns, we recommend that you use this method only if other methods are unsuitable for your site. This method is not available for Windows firewall hosts.

An alternative to using mail, **ftp**, or custom software is to install the store-and-forward software on a "firewall host," a host that can communicate through the firewall. MultiSite synchronization commands can forward data intended for systems on the other side of the firewall to this host.

The software on this host then forwards packets through the firewall to the next hop. To specify the range of port numbers to be used on the host, you can use the environment variables **CLEARCASE_MIN_PORT** and **CLEARCASE_MAX_PORT**. In Figure 24, the hosts that communicate through the firewall are the firewall hosts; they have the MultiSite store-and-forward software installed on them, but not ClearCase software. The replica server hosts have Rational ClearCase and MultiSite installed on them.

Figure 24 Store-and-Forward Configuration



This section describes issues you must consider before installing MultiSite on a firewall host and gives instructions for installation.

Firewall Issues

Before enabling **shipping_server** on a firewall host, consider the following issues:

- Shipping bays can be overfilled.

  Using **shipping_server** on a firewall host enables anyone coming in from the network to fill shipping bays on the local network, on any machine where a **shipping_server** is available. To avoid full disks and the related problems:

  - Ensure that all shipping bays in the local network are on partitions of their own, so that filling the bays does not degrade system performance.
  - Install **shipping_server** only on machines that need it: servers with replicated VOBs and machines used by administrators.

- Packets are susceptible to snooping.

  In normal update packets, version information is not encoded. Therefore, anyone shipping packets across an unsecured network must encrypt the packets. Also, the format of a update packet is not very complicated; a dedicated programmer could figure out the format and create a packet with operations that damage a VOB. Encrypting the data makes this kind of attack much more difficult.

- Other servers can be accessible.

  Allowing **shipping_server** access also allows access to all servers created by the **albd_server**. Because the **albd_server** assigns port numbers in the allowed range to other servers running locally, programs from the outside network can connect to all of those servers. Therefore, the firewall host that runs the **shipping_server** must not run other ClearCase servers.

  If you can specify the ports to which programs can connect and the IP addresses that are allowed to connect, we recommend that you do so. It further limits the possibility that unauthorized machines can breach the firewall. (You specify ports during the firewall configuration process.)

Installing shipping_server on a Firewall

On UNIX, the ClearCase Product Family installation includes an option to install only the **shipping_server** software. Follow the instructions in the *Installation Guide* for the ClearCase Product Family and select only the shipping_server-only option. Do not install ClearCase on the firewall host.

On Windows, there is no installation option for installing only the **shipping_server** software.

Controlling Ports Used by albd_server and shipping_server

The environment variables CLEARCASE_MIN_PORT and CLEARCASE_MAX_PORT specify the range of port numbers that the **albd_server** and **shipping_server** can allocate for communication purposes. When the server needs to assign a port number, it starts with the value of CLEARCASE_MIN_PORT and continues through the range until it reaches CLEARCASE_MAX_PORT. If a port in the range cannot be allocated, the server sleeps and then tries the ports again.

When **shipping_server** detects that the port environment variables are set, it tries to use TCP to make the connection with the **albd_server** on the receiving host. If this connection fails, **shipping_server** tries UDP. Therefore, if you have TCP connectivity, you do not have to enable UDP or open UDP ports on the firewall host.

Running an individual **shipping_server** does not require more than two ports at a time. When there are multiple requests to be sent, **shipping_server** forks. Child processes handle individual requests. The **shipping_server** starts no more than 10 child processes (and starts that many only if there are 10 requests to process simultaneously), so the maximum range is 20 ports. If the range is smaller, it may result in failed attempts, which can be retried later.

Guidelines for Setting Port Values

The value range for CLEARCASE_MIN_PORT is 1024 through 65534, and the value range for CLEARCASE_MAX_PORT is 1025 through 65535. The value of CLEARCASE_MAX_PORT must be greater than the value of CLEARCASE_MIN_PORT.

NOTE: We recommend that you use the range 49152 through 65535, which is the Dynamic/Private Port Range. If you use a value within the Registered Ports range (1024 through 49151), the **shipping.conf** parser prints an informational message.

Specifying Port Values

To specify minimum and maximum port values, set the CLEARCASE_MIN_PORT and CLEARCASE_MAX_PORT environment variables in the following places:

- The **shipping.conf** file on the firewall host. For more information, see the **shipping.conf** reference page.

➤ The **atria_start** script:

a. On the firewall host, edit the file *ccase-home-dir*/**etc/atria_start**.

b. Add the following lines, replacing *min-port* and *max-port* with your minimum and maximum port values. These lines must precede the section that starts the **albd_server**.

```
#
# Set values for minimum and maximum port numbers
#
CLEARCASE_MIN_PORT=min-port
CLEARCASE_MAX_PORT=max-port
export CLEARCASE_MIN_PORT
export CLEARCASE_MAX_PORT
```

6.5 Manual Synchronization

This section describes how to synchronize replicas by entering explicit **syncreplica** commands.

Export Phase

1. **Create an update packet.** At the sending host, use the **syncreplica –export** command with the appropriate transport option.

   If your sites are connected electronically, you can use store-and-forward to send the packet (**–fship**) or place it in a storage bay (**–ship**):

   multitool syncreplica –export –maxsize 1m –fship boston_hub@\dev

```
Generating synchronization packet C:\Program Files\Rational\ClearCase\var
\shipping\ms_ship\outgoing\sync_bangalore_30-Oct-00.14.35.49_2468_1
 - shipping order file is C:\Program Files\Rational\ClearCase\var\shipping
\ms_ship\outgoing\sh_o_sync_bangalore_30-Oct-00.14.35.49_2468_1
Attempting to forward/deliver generated packets...
 -- Forwarded/delivered packet C:\Program Files\Rational\ClearCase\var
\shipping\ms_ship\outgoing\sync_bangalore_30-Oct-00.14.35.49_2468_1
```

   This example uses the **–out** option to save the packet as an output file and includes the **–maxsize** option to divide the logical packet into appropriately sized physical packets. The packet files can then be sent by electronic mail or copied onto diskettes.

multitool syncreplica –export –maxsize 1m –out c:\packets\update1 boston_hub@\dev

```
Generating synchronization packet c:\packets\update1
```

Transport Phase

2. **Send the packets.** Use electronic mail, regular mail, or your preferred delivery method. If you used **syncreplica –export –ship**, invoke **shipping_server** in either of the following ways:

   shipping_server –poll
   shipping_server *shipping-order-pathname*

Import Phase

3. **(If you used diskettes, CD-ROM, tape, or electronic mail) Copy the packet files into a directory.**

4. **Apply the packet.** At the receiving replica, use the **syncreplica –import** command to apply the changes in the packet to the replica.

   This example specifies the **–receive** option; **syncreplica** imports any packets it finds in the incoming shipping directories.

   multitool syncreplica –import –receive

   This example specifies a directory pathname as an argument. **syncreplica –import** looks in this directory for update packets and applies them to the replica on the host.

   multitool syncreplica –import c:\msite\packets

   ```
   Applied sync. packet c:\msite\packets\update1 to VOB \\servo\vobs\dev.vbs
   ```

6.6 Automated Synchronization

You can use MultiSite scripts and utilities to automate all phases of synchronization:

- **Export phase**. A MultiSite export script sends update packets from one or more replicas at the site to one or more siblings.
- **Transport phase**. The *store-and-forward* facility handles packets of any size. You can invoke store-and-forward as part of the export phase, or automate packet transport separately.
- **Import phase**. A MultiSite receipt handler runs whenever a packet is received at a replica, and you can also schedule import of packets to occur periodically.

Use scheduled jobs to automate the export and transport phases, and use receipt handlers or scheduled jobs to automate the import phase. You can run the MultiSite scripts at any time and with any frequency, and you can vary the strategy for different VOBs by using multiple jobs.

By default, the MultiSite synchronization scripts place packets and shipping orders in the **incoming** and **outgoing** directories in the default storage bay, *ccase-home-dir*/**shipping/ms_ship** (UNIX) or *ccase-home-dir***var\shipping\ms_ship** (Windows). This bay is defined in the **shipping.conf** template file on UNIX and the **MultiSite Control Panel** on Windows.

The MultiSite scripts log their activity to files in the **/var/adm/atria/log/sync_logs** directory on UNIX and the *ccase-home-dir***var\log** directory on Windows.

Using the ClearCase Scheduler

ClearCase installation adds three preconfigured jobs to the scheduler: **Daily MultiSite Export**, **Daily MultiSite Shipping Poll**, and **Daily MultiSite Receive**. These jobs use the predefined MultiSite tasks: **MultiSite Sync Export** and **MultiSite Sync Receive**.

These jobs are disabled; to enable them, use the **schedule –edit –schedule** command or the graphical interface (Windows) and set the run times and other parameters appropriately:

- (Using **cleartool schedule**) Delete the line `Job.Schedule.LastDate: StartDate` and set the value of `Job.NotifyInfo.Recipients` to the appropriate user names.
- (Using the scheduler graphical interface) On the **Schedule** tab, set the **Run** parameters to the appropriate values. On the **Settings** tab, in the **Notifications** section, change the value of **Recipients** to the appropriate user names.

NOTE: If you decide to use receipt handlers (see *Import Phase* on page 107), you do not need to enable the **Daily MultiSite Receive** job.

For information about creating new tasks and jobs and the prerequisites for using the scheduler, see the **cleartool schedule** reference page in the *Command Reference* and the *Administrator's Guide* for Rational ClearCase.

Export Phase

The script **sync_export_list** creates update packets. You can select the replicas to be updated, configure the script to send the packets immediately or place them in storage bays, and set other shipping options. For more information on the shipping options, see the **sync_export_list** reference page.

This job runs **sync_export_list** to generate and send updates to all other replicas in the VOB family at midnight local time:

```
Job.Begin
     Job.Id: 17
     Job.Name: "Sync Export Force ALL"
     Job.Description.Begin:
Every midnight, for each replica on this host, export update packets to all
sibling replicas.
     Job.Description.End:
     Job.Schedule.Daily.Frequency: 1
     Job.Schedule.FirstStartTime: 00:00:00
     Job.DeleteWhenCompleted: FALSE
     Job.Task: 13
     Job.Args: -quiet 1 -all
Job.End
```

To put the packets in a storage bay, use the **–ship** option. Packets in storage bays are sent by the **shipping_server**. For example, this job runs **sync_export_list** to generate an update every day at 21:00 local time:

```
Job.Begin
     Job.Id: 18
     Job.Name: "Sync Export Store ALL"
     Job.Description.Begin:
Every night at 9PM, for each replica on this host, generate update packets for
all sibling replicas and store the packets in the storage bay.
     Job.Description.End:
     Job.Schedule.Daily.Frequency: 1
     Job.Schedule.FirstStartTime: 21:00:00
     Job.DeleteWhenCompleted: FALSE
     Job.Task: 13
     Job.Args: -quiet 1 -ship -all
Job.End
```

See *Transport Phase* for information about running **shipping_server**.

Transport Phase

If **sync_export_list** or **syncreplica** puts packets in storage bays (**–ship** option), you must run **shipping_server** to process these packets. If you do not use **–ship**, but want to implement a retry-on-failure capability, you must schedule regular invocations of **shipping_server**. The **shipping_server** attempts to retransmit any outgoing packets that remain in any of the storage bays because one or more previous attempts have failed.

With the **–poll** option, **sync_export_list** invokes **shipping_server –poll** to process shipping orders located in all storage bays defined in the **shipping.conf** file (UNIX) or in the **MultiSite Control Panel** (Windows).

For example, this job invokes **shipping_server** every day at 04:00 local time:

```
Job.Begin
     Job.Id: 19
     Job.Name: "Shipping Server Poll"
     Job.Description.Begin:
Every night at 4AM, run the shipping server to send any outstanding orders.
     Job.Description.End:
     Job.Schedule.Daily.Frequency: 1
     Job.Schedule.FirstStartTime: 04:00:00
     Job.DeleteWhenCompleted: FALSE
     Job.Task: 13
     Job.Args: -quiet 1 -poll
Job.End
```

The following job implements a more aggressive retry-on-failure capability:

```
Job.Begin
     Job.Id: 20
     Job.Name: "Shipping Server Poll"
     Job.Description.Begin:
Every half hour from midnight to 4AM, run the shipping server to send any
outstanding orders.
     Job.Description.End:
     Job.Schedule.Daily.Frequency: 1
     Job.Schedule.FirstStartTime: 00:00:00
     Job.Schedule.StartTimeRestartFrequency: 00:30:00
     Job.Schedule.LastStartTime: 04:00:00
     Job.DeleteWhenCompleted: FALSE
     Job.Task: 13
     Job.Args: -quiet 1 -poll
Job.End
```

Import Phase

To automate packet import, use one of the methods described in Table 11.

Table 11 Import Methods

Import method	Description	Advantages	Disadvantages
Receipt handlers	When a packet is received, the receipt handler imports it.	Packets are imported immediately.	Constant load on VOB server.
Scheduled jobs	When a packet is received at a replica, it is stored in a shipping bay. When the scheduled job runs, the packet is imported.	You can schedule import to minimize server load.	Changes are not applied to the VOB immediately.
Receipt handlers and scheduled jobs	See descriptions above.	You can use scheduled jobs to implement a retry-on-failure capability. For example, if packets are delivered out of order and the receipt handler cannot import them, the job retries the import.	

Defining Receipt Handlers

On UNIX:

You can define receipt handlers in the **shipping.conf** file for different shipping classes. By default, no receipt handler is defined, but you can specify the **sync_receive** script as a receipt handler in the **shipping.conf** file:

```
RECEIPT-HANDLER -default /usr/atria/config/scheduler/tasks/sync_receive
```

For details about defining receipt handler entries, see the section *RECEIPT HANDLER* in the **shipping.conf** reference page.

On Windows:

You can define receipt handlers in the **MultiSite Control Panel** for different shipping classes. By default, no receipt handler is defined, but you can specify *ccase-home-dir***\config\scheduler\tasks\sync_receive.bat** in the **MultiSite Control Panel**. To customize **sync_receive.bat**, copy it to a directory outside the ClearCase installation directory, customize it, and specify it in the MultiSite Control Panel.

For details about defining receipt handler entries, see the section *Receipt Handler Path* in the **MultiSite Control Panel** reference page.

Scheduling Import Jobs

The script **sync_receive** imports update packets. For more information on **sync_receive** options, see the **sync_receive** reference page.

This job runs **sync_receive** to import all packets in the incoming shipping bays of the current host at midnight local time:

```
Job.Begin
     Job.Id: 15
     Job.Name: "Sync Import ALL"
     Job.Description.Begin:
Every midnight, import all update packets in incoming bays.
     Job.Description.End:
     Job.Schedule.Daily.Frequency: 1
     Job.Schedule.FirstStartTime: 00:00:00
     Job.DeleteWhenCompleted: FALSE
     Job.Task: 14
Job.End
```

6.7 Listing Synchronization History

The **lshistory** command and the History Browser list the history of a replica, including synchronization information. For more information, see *Listing the Synchronization History of a Replica* on page 113.

6.8 Synchronizing More Efficiently

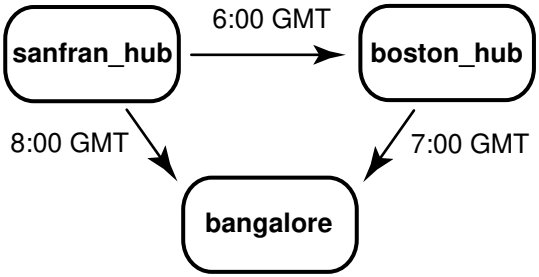
You can configure synchronization updates to send only the necessary operations to another replica. Although sending an operation multiple times does no harm, packet creation and transmission is more efficient if you exclude operations that have already been imported at the receiving replica.

The **chepoch –actual** and **sync_export_list –update** commands contact a remote replica and update your current replica's record of the state of the remote replica. The primary use of these commands is to resend lost packets, but you can also use them to increase synchronization efficiency. However, depending on your synchronization pattern and schedule, these commands can decrease efficiency. The following sections describe two examples: one in which efficiency is increased, and one in which it is decreased.

Example of Increased Efficiency

You have three replicas in a VOB family, and a subset of the synchronization pattern and schedule is shown in Figure 25. All replicas use receipt handlers, so incoming packets are imported immediately. First, replica **sanfran_hub** sends a packet to replica **boston_hub**. Next, replica **boston_hub** sends a packet to replica **bangalore**. This packet includes operations from replica **sanfran_hub**.

Figure 25 Partial Synchronization Export Pattern and Schedule for Three Replicas



At 8:00 GMT, replica **sanfran_hub** sends a packet to replica **bangalore**. This packet contains operations originating at replica **sanfran_hub** that **bangalore** has already received from replica **boston_hub**. In this case, you should use the command **chepoch –actual bangalore** at replica **sanfran_hub** before generating an update packet for **bangalore**. When you generate the packet, the operations already imported at bangalore will be excluded from the packet.

Example of Decreased Efficiency

In this example, two replicas in a VOB family, **sanfran_hub** and **sydney**, exchange update packets every fifteen minutes. At some point during the day, packets may start accumulating at one of the replicas because the imports are taking a long time. For example, there is a lot of development activity in the **sydney** VOB and there are four packets waiting to be imported.

In this case, if you run **chepoch –actual** at **sanfran_hub** before generating a packet for **sydney**, the update packet will contain all the operations that are already in the packets waiting to be imported at sydney.

7 Managing Replicas

This chapter describes how to manage existing replicas, including how to delete a replica. For information about creating a replica, see Chapter 4, *Creating Replicas*. For information about enabling requests for mastership in a replica, see Chapter 9, *Implementing Requests for Mastership*.

7.1 Displaying Properties of a Replica

The **describe** command, which is available in **cleartool** and **multitool**, displays the properties of a replica. Use the **–fmt** option to customize the output from the command. See the **fmt_ccase** reference page in the *Command Reference*.

For example, to display the name, master replica, and replica host of a replica:

cleartool describe –fmt "%n\t%[master]p\t%[replica_host]p\n" \ replica:boston_hub@/vobs/dev

```
boston_hub            boston_hub@/vobs/dev            minuteman
```

You can also display properties of a replica by using a *replica-uuid-selector* instead of a *replica-selector* argument. For example (note that the *replica-uuid-selector* must be entered on a single line):

cleartool describe –fmt "%n\n%[master]p\n%[replica_host]p\n" \ oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7@replicauuid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7

```
boston_hub
boston_hub@/vobs/dev
minuteman
```

On Windows, the Properties Browser displays the properties of a replica. Open the Properties Browser in one of the following ways:

- From Windows Explorer:

  a. Navigate to the VOB.

  b. Right-click the VOB and click **ClearCase > Properties of VOB**.

  c. Click the **Replicas** tab.

  d. Select the replica and click **Replica Properties**.

- From ClearCase Administration Console:

  a. Navigate to **All VOBs**.

  b. Click **View > List**.

  c. Right-click the VOB and click **Properties**.

  d. Click the **Replicas** tab.

  e. Select the replica and click **Replica Properties**.

- From a command prompt:

  cleardescribe replica:*replica-selector*

  cleartool describe –graphical replica:*replica-selector*

  For example:

  cleardescribe replica:boston_hub@\dev

  cleartool describe –graphical replica:sanfran_hub@\tests

7.2 Listing the Synchronization History of a Replica

The **lshistory** command and the History Browser (**lshistory –graphical**) list the synchronization history of a replica. The output differs for your current replica and its sibling replicas:

- When you list the history of your current replica, the output includes import events.
- When you list the history of a sibling replica, the output includes export events from your current replica to the sibling replica.

To list the import history of your current replica (**boston_hub**):

cleartool lshistory replica:boston_hub@/vobs/dev

```
17-Aug.11:05 susan import sync from replica "sanfran_hub" to replica
"boston_hub"
  "Imported synchronization information from replica "sanfran_hub".
  Row at import was: boston_hub=34 sanfran_hub=0"
```

To list all exports from your current replica to the **sanfran_hub** replica:

cleartool lshistory replica:sanfran_hub@/vobs/dev

```
17-Aug.11:11 susan export sync from replica "boston_hub" to replica
"sanfran_hub"
  "Exported synchronization information for replica "sanfran_hub".
  Row at export was: boston_hub=34 sanfran_hub=10"
17-Aug.10:54 susan export sync from replica "boston_hub" to replica
"sanfran_hub"
  "Exported synchronization information for replica "sanfran_hub".
  Row at export was: boston_hub=0 sanfran_hub=0"
16-Aug.09:49 susan create replica "sanfran_hub"
```

7.3 Changing the Host Name for a Replica

When you move a replica's storage directory to a different host, or when you rename a replica's host, you must update the host name in the replica's VOB database. The database keeps track of the hosts on which the replicas in a VOB family reside so that the store-and-forward facility can determine how to route updates to the replicas.

To change the host name, use the **chreplica** command or the Properties Browser (Windows). The change is not propagated to other replicas in the VOB family until you export an update packet

from the current replica and the packet is imported at the other replicas. For restrictions, see the **chreplica** reference page.

To change a host name using the **chreplica** command:

multitool chreplica –host mardelplata buenosaires@/vobs/dev

```
Updated replica information for "buenosaires".
```

To change a host name using the Properties Browser:

1. Display properties of the replica. See *Displaying Properties of a Replica* on page 111.
2. On the **General** tab, type the new host name in the **Host** box.
3. Click **OK** or **Apply**.

7.4 Changing Ownership Preservation

Any subset of replicas in a VOB family can be *ownership-preserving*. Within this group of replicas, the owner, group, and access mode of an object are kept the same across all the replicas. Adding a replica to or deleting it from the group has no immediate effect on the replica's objects. However, future changes to object permissions are propagated among all of the ownership-preserving replicas in the VOB family.

NOTE: On UNIX, maintaining ownership preservation across sites is possible only if all sites support the same user-group accounts. On Windows, ownership modes (*UID*s and *GID*s) are not consistent across domains. Therefore, if all replicas in a VOB family are not in the same Windows domain, the entire set of replicas cannot be ownership-preserving. You can maintain ownership preservation on a subset of replicas in the same domain. In a mixed environment, you cannot maintain ownership preservation on the entire set of replicas. For more information, see *Element Ownership and Ownership Preservation* on page 4.

The most common change is to convert a replica from ownership-preserving to non-ownership-preserving. For example, if a replica was created incorrectly as ownership-preserving, you may need to change it. You can change a replica from non-ownership-preserving to ownership-preserving. The replica will receive future changes to ownership information, but the original ownership information is not restored.

To change a replica's ownership-preserving property:

1. At the master replica, change the replica property.

   On UNIX or Windows, you can use the **chreplica** command to change this property:

   - To change from non-preserving to preserving:

     multitool chreplica –preserve boston_hub@/vobs/dev
     `Updated replica information for "boston_hub".`

   - To change from preserving to non-preserving:

     multitool chreplica –npreserve boston_hub@/vobs/dev
     `Updated replica information for "boston_hub".`

   On Windows, you can use the Properties Browser to change this property:

   a. Display properties of the replica. See *Displaying Properties of a Replica* on page 111.

   b. To change from non-preserving to preserving, select the **Ownership-preserving** check box. To change from preserving to non-preserving, clear the **Ownership-preserving** check box.

   c. Click **OK** or **Apply**.

   See the **chreplica** reference page for restrictions.

2. If the changed replica is not self-mastering, export an update packet from the master replica to the changed replica.

3. At the changed replica, import the update packet. If the import fails because the VOB group lists are different, use the **cleartool protectvob** command to change the group list for the importing VOB replica, and then try the import again.

   If the import succeeds, you can use the **protectvob** command to delete the group you added.

4. (If the replica was changed to non-ownership-preserving) At the changed replica, use the **cleartool protect** command to change the ownership of all elements in the replica to the VOB owner at your site.

   cleartool protect –chown vobowner –chgrp vobgrp –recurse /vobs/dev

7.5 Setting the Connectivity Property

To indicate whether a sibling replica has IP connectivity to your current replica, use the **chreplica** command with the **–isconnected** or **–nconnected** option. This property is stored locally and is checked when you enter a command that requires connectivity to a sibling replica (for example, **lsepoch –actual** or **chepoch –actual**). The command fails quickly if the sibling replica is marked as not connected.

You can also use the Properties Browser on Windows. When you display properties of a sibling replica, the **General** tab indicates whether the replica has IP connectivity to the current replica. You can change this property by setting or clearing the check box.

To use the **chreplica** command to set the connectivity property to connected for the **sanfran_hub** replica:

multitool chreplica –isconnected sanfran_hub

```
Updated replica information for "sanfran_hub".
```

multitool describe replica:sanfran_hub

```
replica "sanfran-hub"
...
  connectivity: connected
```

For more information, see the **chreplica** reference page.

7.6 Renaming a Replica

To change the name of a replica, use the **rename** command or the Properties Browser (Windows). When you rename a replica, the change is made immediately at the current replica. The change is not propagated to other replicas in the VOB family until you export an update packet from the current replica and the packet is imported at the other replicas.

You must make the change at the replica's master replica. For other restrictions, see the **rename** reference page in the *Command Reference*.

To rename a replica using the **rename** command:

multitool rename –c "site name" replica:original@/vobs/dev replica:boston_hub@/vobs/dev

```
Renamed replica from "original" to "boston_hub".
```

To rename a replica using the Properties Browser:

1. Display properties of the replica. See *Displaying Properties of a Replica* on page 111.
2. Enter a new value in the **Name** box.
3. Click **OK** or **Apply**.

7.7 Moving a Replica

See the information on moving a VOB in the *Administrator's Guide* for Rational ClearCase.

There are some special considerations when you move a replicated VOB:

- Make sure Rational ClearCase MultiSite is installed on the new VOB server host.
- If you automated the synchronization process on the old host, you must set up synchronization export and import scripts on the new VOB server host.
- After moving the VOB replica, change the host name associated with the replica by using **multitool chreplica –host**. You must enter this command at the master replica of the replica you moved.
- After moving the VOB replica, export update packets to all sibling replicas.

7.8 Changing Mastership of a Replica

When you create a new replica, its replica object is mastered by the replica at which you enter the **mkreplica –export** command. Mastership of the replica object affects replica-modification activities (renaming the replica, changing its properties, or deleting it). You must perform these activities at the replica that masters the replica object.

A self-mastering replica masters its own replica object. A replica must be self-mastering for you to perform some administrative operations on it (for example, raising the feature level). If each site has its own MultiSite administrator, having self-mastering replicas simplifies replica maintenance because each replica can be maintained at its own site. However, you may want to master all replica objects at a hub replica. In this case, you can transfer mastership to individual

sites at the request of the site administrator, and then transfer mastership back to the hub replica after the administrative operation has been completed.

To transfer mastership of a replica object:

1. Determine which replica masters the replica object, and the host name of the replica's VOB server:

   multitool describe replica:sanfran_hub@/vobs/dev
   ```
   replica "sanfran_hub"
   created 16-Aug-00.09:49:36 by Susan Goechs (susan.user@minuteman)
   replica type: unfiltered
   master replica: boston_hub@/vobs/dev
   owner: susan
   group: user
   host: "goldengate"
   ...
   ```

2. At the master replica, enter a **chmaster** command:

   MINUTEMAN% **multitool chmaster –c "make sanfran_hub replica self-mastering" \ sanfran_hub@/vobs/dev replica:sanfran_hub@/vobs/dev**
   ```
   Changed mastership of replica "sanfran_hub" to "sanfran_hub@/vobs/dev"
   ```

3. At the old master replica, export an update packet to the new master replica:

   MINUTEMAN% **multitool syncreplica –export –fship sanfran_hub@/vobs/dev**
   ```
   Generating synchronization packet
   /usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_16-Aug-00.16.15.57_63
   89_1
   - shipping order file is
   /usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_16-Aug-00.16.15.
   57_6389_1
   Attempting to forward/deliver generated packets...
   -- Forwarded/delivered packet
   /usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_16-Aug-00.16.15.57_63
   89_1
   ```

4. At the new master replica, import the packet:

   GOLDENGATE% **multitool syncreplica –import –receive**
   ```
   Applied sync. packet
   /usr/atria/shipping/ms_ship/incoming/sync_boston_hub_16-Aug-00.16.15.57_63
   89_1 to VOB /net/goldengate/vobstg/dev.vbs
   ```

5. At the new master replica, verify that mastership has been received:

```
GOLDENGATE% multitool describe replica:sanfran_hub@/vobs/dev
replica "sanfran_hub"
created 16-Aug-00.09:49:36 by Susan Goechs (susan.user@minuteman)
replica type: unfiltered
master replica: sanfran_hub@/vobs/dev
...
```

7.9 Deleting a Replica

This section describes how to remove a replica. You must complete all steps; if you do not, synchronization and mastership problems can occur in other replicas in the VOB family.

NOTE: If a VOB replica is deleted mistakenly and you want to restore it from backup, see *Restoring and Replacing Replicas* on page 190. If a VOB replica's storage directory is lost and there is no backup, see *Cleaning Up from Accidental Deletion of a Replica* on page 196.

In this scenario, the replica **tokyo** in the VOB family **\tests** is being removed.

1. At the site of the replica to be removed, complete all development work in the replica and use **lscheckout** or the Find Checkouts tool (Windows) to verify that all checkouts are resolved in the replica to be removed:

   ```
   SHINJUKU> cleartool lscheckout –all \tests
   (no output means no checkouts)
   ```

2. Transfer mastership of all objects to another replica.

   At the site of the replica to be removed, transfer mastership of all objects mastered by the replica to another replica. If the replica to be removed is not self-mastering, transfer mastership to its master replica. If the replica is self-mastering, you can choose any replica.

   In this example, the administrator determines which replica masters **tokyo**, and then transfers mastership to the master replica (in this example, **sanfran_hub**):

   ```
   SHINJUKU> cleartool describe –fmt "%[master]p\n" replica:tokyo@\tests
   sanfran_hub@\tests

   SHINJUKU> multitool chmaster –all –long sanfran_hub@\tests
   Changed mastership of versioned object base \tests
   ...
   Changed mastership of all objects
   ```

The replica that receives the mastership can later transfer mastership to other replicas.

If mastership is not transferred for all objects, you must fix the problem and reenter the **chmaster –all –long** command. For an example, see *Transferring Mastership of All Objects Mastered by a Replica* on page 135. If there are problems you cannot fix, another replica can recover from the error by assuming mastership of the objects. For a description of this procedure, see *Cleaning Up from Accidental Deletion of a Replica* on page 196.

3. Export and send an update packet from the replica to be removed.

   The replica to be removed must send its final changes, including checkins and mastership changes, to the replica receiving mastership. The replica to be removed can broadcast its final changes to all other replicas, but it must update its master replica (**sanfran_hub** in this example).

   `SHINJUKU>` **multitool syncreplica –export –fship sanfran_hub@\tests**

4. Import the update packet at the replica that is (or will become) the master of the replica to be removed.

   `GOLDENGATE%` **multitool syncreplica –import –receive**

5. At the master replica, remove the replica.

   `GOLDENGATE%` **multitool rmreplica tokyo@/vobs/tests**

6. At the master replica, export and send an update packet to the remaining replicas in the VOB family.

   This update packet notifies the other replicas of the replica removal.

   `GOLDENGATE%` **multitool syncreplica –export –fship <***replica names***>**

7. At the removed replica, remove the VOB storage directory of the removed replica.

   `SHINJUKU>` **cleartool rmvob \\shinjuku\vobs\tests.vbs**
   `Remove versioned object base "\\shinjuku\vobs\tests.vbs"?  [no]` **yes**
   `Removed versioned object base "\\shinjuku\vobs\tests.vbs".`

If you decommission and remove all replicas, the one remaining VOB replica is a regular VOB, and developers no longer need a MultiSite license to access it.

8 Managing Mastership

This chapter describes how to manage the mastership of ClearCase objects in a VOB replica, using the following commands:

- **describe**
- **lsmaster**
- **mkelem –master**
- **chmaster**

The **mkelem** command is a **cleartool** subcommand. The other commands listed above are **cleartool** and **multitool** subcommands. For more information on the commands, see their reference pages in this manual or in the *Command Reference*.

On Windows, you can use the **describe** and **chmaster** commands or the Properties Browser to display and change mastership.

The **reqmaster** command requests mastership of branches and branch types and sets controls for mastership requests. On Windows, you can use the Request Mastership graphical interface and the Properties Browser to request mastership and set controls. Use of these interfaces is described in Chapter 9, *Implementing Requests for Mastership*.

NOTE: Before reading this chapter, you should read the information in *Enabling Independent VOB Development: Mastership* on page 7.

8.1 Listing an Object's Master Replica

To list an object's master replica, use one of the following methods:

Mastership page in the Properties Browser (Windows)	This page lists the object's master replica.
cleartool describe *object-selector*	This command lists general information about the object, including its master replica.
cleartool describe –fmt "%[master]p\n" *object-selector*	This command lists only the master replica of the object. For more information on using the **–fmt** option, see the **fmt_ccase** reference page in the *Command Reference*.

Command examples:

➤ To list a replica object's master replica:

cleartool describe replica:boston_hub@\dev

```
replica "boston_hub"
created 15-Aug-00.14:19:03 by susan.user
replica type: unfiltered
master replica: boston_hub@\dev
...
```

➤ To list an element's master replica:

cleartool describe –fmt "%n\t%[master]p\n" cmdsyn.c@@

```
cmdsyn.c@@      bangalore@/vobs/dev
```

➤ To list a type object's master replica:

cleartool describe lbtype:V1.0@/vobs/dev

```
label type "V1.0"
  created 25-Aug-00.12:14:52 by Susan Goechs (susan.user@minuteman)
  master replica: boston_hub@/vobs/dev
...
```

➤ To list a branch's master replica:

cleartool describe –fmt "%n\t%[master]p\n" cmdsyn.c@@\main\v3_bugfix

```
cmdsyn.c@@\main\v3_bugfix boston_hub@\dev
```

8.2 Listing Objects Mastered by a Replica

The **lsmaster** command lists the objects mastered by a replica. The command uses the information at your current replica unless you use the **–inreplicas** option, which retrieves information from sibling replicas.

➤ To list all objects mastered by the current replica (**boston_hub**):

multitool lsmaster boston_hub@/vobs/dev

➤ To list all label types mastered by replica **sanfran_hub**, using information in the current replica:

multitool lsmaster –kind lbtype sanfran_hub@/vobs/dev

➤ To display information from all replicas in the VOB family about the objects mastered by replica **bangalore**:

multitool lsmaster –inreplicas –all bangalore@\tests

For more information and examples, see the **lsmaster** reference page.

8.3 Listing the History of Mastership Changes for an Object

The **lshistory –minor** command and the History Browser (with the **Include minor events** toggle selected) list mastership changes for an object. For example, to list the history of a replica:

cleartool lshistory –minor replica:bangalore@/vobs/dev

```
20-Sep.17:43   susan      change master to "bangalore" of replica "bangalore"
"Changed master replica from "boston_hub" to "bangalore"."
```

8.4 Displaying Mastership Request Settings

The mastership request setting controls whether developers at other sites can request mastership of branches or branch types mastered by the replica. The **describe** command and (on Windows) the Mastership page in the Properties Browser display mastership request settings for replicas, branch types, and branches. For more information on mastership requests, see Chapter 9, *Implementing Requests for Mastership*.

8.5 Assigning Branch Mastership During Element Creation

By default, when you create an element in a replicated VOB, mastership of the branch **main** is assigned to the replica that masters the branch type **main**. If this replica is not your current replica, you cannot create new versions on the main branch. Also, if your config spec contains **mkbranch** rules and your current replica does not master the branch types, the branches cannot be created during element creation.

To assign mastership of a new element's **main** branch, and all other branches created during element creation, to your current replica, do one of the following:

- Use the command **cleartool mkelem –master**.
- (Windows) In the Add to Source Control dialog box, select **Make current replica the master of all newly created branches**.

For example, suppose your view has the following config spec:

```
element * CHECKEDOUT
element * .../v1.0_bugfix/LATEST
element * V1.0 -mkbranch v1.0_bugfix
```

Use the following procedure to assign mastership of new branches to your current replica:

1. Create a new element with **mkelem –master** and check out the file:

   cleartool mkelem –master –c "adding comments" cmdsyn.c
   ```
   Created element "cmdsyn.c" (type "text_file").
   Created branch "v1.0_bugfix" from "cmdsyn.c" version "/main/0".
   Note: Branch "v1.0_bugfix" explicitly mastered by replica "boston_hub".
   Branch type "v1.0_bugfix" mastered by replica "sanfran_hub".
   Checked out "cmdsyn.c" from version "/main/v1.0_bugfix/0".
   ```

2. Use the **describe** command to confirm that the new branches are mastered by your current replica:

   cleartool describe cmdsyn.c@@/main cmdsyn.c@@/main/v1.0_bugfix
   ```
   branch "cmdsyn.c@@/main"
     created 02-Sep-00.13:17:21 by Gail Smith (gail.user@boston30)
     branch type: main
     master replica: boston_hub@/vobs/dev
   ...
   branch "cmdsyn.c@@/main/v1.0_bugfix"
     created 02-Dec-00.13:17:21 by Gail Smith (gail.user@boston30)
     branch type: v1.0_bugfix
     master replica: boston_hub@/vobs/dev
   ...
   ```

If you make your current replica the master of newly created branches, but do not check out the file (that is, you use the **–nco** option), only the main branch is mastered by your current replica, because it is the only branch that is created. For example:

1. Create a new element with **mkelem –nco –master**:

   cleartool mkelem –nco –master –c "adding comments" cmdsyn.c
   ```
   cleartool: Warning: Moved private data from "cmdsyn.c" to "cmdsyn.c.keep"
   so it won't eclipse element.
   Created element "cmdsyn.c" (type "text_file").
   ```

2. Use the **describe** command to confirm that the **main** branch is mastered by your current replica:

   cleartool describe cmdsyn.c@@/main
   ```
   branch "cmdsyn.c@@/main"
   created 02-Sep-00.13:21:21 by Gail Smith (gail.user@boston30)
   branch type: main
   master replica: boston_hub@/vobs/dev
   ...
   ```

3. List the element's history to confirm that no other branches except **main** were created:

   cleartool lshistory cmdsyn.c
   ```
   02-Sep.13:21   gail      create version "cmdsyn.c@@/main/0"
   02-Sep.13:21   gail      create branch "cmdsyn.c@@/main"
   02-Sep.13:21   gail      create file element "cmdsyn.c@@"
   ```

8.6 Changing Mastership

When you create an object in a replicated VOB, your current replica is the new object's master. You can transfer mastership of the object to another replica, using the **chmaster** command or the Properties Browser (Windows). Some examples of when this is appropriate:

- If responsibility for product integration is shifted to a different site, you must transfer mastership of the integration branch types or branches to the new site.
- Before you decommission a replica, you must transfer mastership of each object mastered by that replica to one of the remaining replicas. (See *Deleting a Replica* on page 119.)

Mastership changes are communicated among replicas by the standard synchronization mechanism. The general procedure for changing mastership is as follows:

1. Change mastership of one or more objects to another replica or request mastership of a branch or branch type.
2. Export and send an update packet from the old master replica to the new master replica. (The **reqmaster** command does this automatically.)
3. Import the update packet at the new master replica.

Until the update packet containing the mastership change is imported at the new master replica, mastership is "in the packet" and the replicas in the VOB family have different information about which replica masters the object.

For example, the administrator at the **sanfran_hub** replica transfers mastership of the **bugfix** branch to the **bangalore** replica, and then exports an update packet. At this point:

- The **sanfran_hub** replica considers the branch to be mastered by **bangalore**.
- The **bangalore** replica considers the branch to be mastered by **sanfran_hub**.
- No one can create new versions on the branch.

When you complete the mastership transfer by importing the update packet at **bangalore**, developers at **bangalore** are able to create new versions on the branch.

Notes on mastership changes:

- The **chmaster** command requires a view context.
- If your VOB family includes any read-only or one-way replicas (replicas that import update packets but do not export them), be careful about transferring mastership to these replicas.

After you give mastership of an object to a read-only or one-way replica, you cannot change the object's mastership again unless you change the VOB family's synchronization pattern.

- You cannot undo a mastership change made at your site by making the opposite change at your site. See *Fixing an Accidental Mastership Change* on page 137.
- You can use triggers to prevent developers from changing mastership of objects. For more information, see *Implementing Project Development Policies* in *Managing Software Projects*.

The following sections describe how to change mastership of ClearCase objects. These procedures use the command line. For information about using the Properties Browser on Windows to transfer mastership of a ClearCase object, see the MultiSite online help:

1. Click **Start > Programs > Rational ClearCase Administration > MultiSite Help**.
2. On the **Contents** tab of the Help Contents Window, click **Administrator Tasks > To change mastership of a ClearCase object**.

Transferring Mastership of a Type Object

When you create a type object, it is mastered by the replica where you create it. Except for elements, instances of an unshared type can be created only at the master replica. Elements can be created at any replica, regardless of which replica masters the element type. Instances of shared types can be created at any replica (for more information, see *Type Object Mastership* on page 14).

NOTE: If you transfer mastership of a branch type to another replica, mastership of explicitly mastered branches of that type is not transferred, even if the same replica masters the branch type and the branch. To give such branches default mastership, see the procedure in *Removing Explicit Mastership of a Branch* on page 133.

To transfer mastership of a type object:

1. Determine which replica masters the type object:

   multitool describe lbtype:TOKYO_BASE@/vobs/dev
   ```
   label type "TOKYO_BASE"
   created 15-Aug-00.14:20:26 by Susan Goechs (susan.user@minuteman)
   master replica: boston_hub@/vobs/dev
   ...
   ```

2. At the master replica, enter a **chmaster** command:

```
MINUTEMAN% multitool chmaster –c "transfer to sanfran_hub" \
sanfran_hub@/vobs/dev lbtype:TOKYO_BASE@/vobs/dev
Changed mastership of label type "TOKYO_BASE" to "sanfran_hub@/vobs/dev"
```

3. At the old master replica, export an update packet to the new master replica's site:

```
MINUTEMAN% multitool syncreplica –export –fship sanfran_hub@/vobs/dev
Generating synchronization packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_26-Aug-00.18.15.57_74
30_1
- shipping order file is
/usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_26-Aug-00.18.15.
57_7430_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_26-Aug-00.18.15.57_74
30_1
```

4. At the new master replica, import the packet:

```
BAGUETTE% multitool syncreplica –import –receive
Applied sync. packet
/usr/atria/shipping/ms_ship/incoming/sync_boston_hub_26-Aug-00.18.15.57_74
30_1 to VOB /net/goldengate/vobstg/dev.vbs
```

5. At the new master replica, verify that mastership has been received:

```
BAGUETTE% multitool describe lbtype:TOKYO_BASE@/vobs/dev
label type "TOKYO_BASE"
created 15-Aug-00.14:20:26 by Susan Goechs (susan.user@minuteman)
master replica: sanfran_hub@/vobs/dev
...
```

Transferring Mastership of a Replica Object

When you create a new replica, its replica object is mastered by the replica at which you enter the **mkreplica –export** command. Mastership of the replica object affects replica-modification activities (renaming the replica, changing its properties, or deleting it). You must perform these activities at the replica that masters the replica object.

A self-mastering replica masters its own replica object. A replica must be self-mastering for you to perform some administrative operations on it (for example, raising the feature level). If each site has its own MultiSite administrator, having self-mastering replicas simplifies replica maintenance because each replica can be maintained at its own site. However, you may want to master all replica objects at a hub replica. In this case, you can transfer mastership to individual sites at the request of the site administrator, and then transfer mastership back to the hub replica after the administrative operation has been completed.

To transfer mastership of a replica object:

1. Determine which replica masters the replica object, and the host name of the replica's VOB server:

   multitool describe replica:sanfran_hub@/vobs/dev
   ```
   replica "sanfran_hub"
   created 16-Aug-00.09:49:36 by Susan Goechs (susan.user@minuteman)
   replica type: unfiltered
   master replica: boston_hub@/vobs/dev
   owner: susan
   group: user
   host: "goldengate"
   ...
   ```

2. At the master replica, enter a **chmaster** command:

   MINUTEMAN% **multitool chmaster –c "make sanfran_hub replica self-mastering" \ sanfran_hub@/vobs/dev replica:sanfran_hub@/vobs/dev**
   ```
   Changed mastership of replica "sanfran_hub" to "sanfran_hub@/vobs/dev"
   ```

3. At the old master replica, export an update packet to the new master replica:

   MINUTEMAN% **multitool syncreplica –export –fship sanfran_hub@/vobs/dev**
   ```
   Generating synchronization packet
   /usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_16-Aug-00.16.15.57_63
   89_1
   - shipping order file is
   /usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_16-Aug-00.16.15.
   57_6389_1
   Attempting to forward/deliver generated packets...
   -- Forwarded/delivered packet
   /usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_16-Aug-00.16.15.57_63
   89_1
   ```

4. At the new master replica, import the packet:

```
GOLDENGATE% multitool syncreplica –import –receive
Applied sync. packet
/usr/atria/shipping/ms_ship/incoming/sync_boston_hub_16-Aug-00.16.15.57_63
89_1 to VOB /net/goldengate/vobstg/dev.vbs
```

5. At the new master replica, verify that mastership has been received:

```
GOLDENGATE% multitool describe replica:sanfran_hub@/vobs/dev
replica "sanfran_hub"
created 16-Aug-00.09:49:36 by Susan Goechs (susan.user@minuteman)
replica type: unfiltered
master replica: sanfran_hub@/vobs/dev
...
```

Transferring Mastership of a VOB

When you replicate a VOB for the first time, the VOB is mastered by the original replica. You must perform the following operations at the VOB's master replica:

- Changing protections on the VOB (for ownership-preserving replicas).
- Locking the VOB with the obsolete option.

To transfer mastership of a VOB to another replica, follow these steps:

1. At the master replica, enter a **chmaster** command:

```
MINUTEMAN% multitool chmaster sanfran_hub vob:/vobs/dev
Changed mastership of versioned object base "/vobs/dev" to "sanfran_hub".
```

2. At the old master replica, export an update packet to the new master replica's site:

```
MINUTEMAN% multitool syncreplica –export –fship sanfran_hub@/vobs/dev
Generating synchronization packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_20-Sep-00.17.35.45_22
513_1
- shipping order file is
/usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_20-Sep-00.17.35.
45_22513_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_20-Sep-00.17.35.45_22
513_1
```

3. At the new master replica, import the packet:

```
GOLDENGATE% multitool syncreplica –import –receive
Applied sync. packet
/usr/atria/shipping/ms_ship/incoming/sync_boston_hub_20-Sep-00.17.35.45_22
513_1 to VOB /net/goldengate/vobstg/dev.vbs
```

4. At the new master replica, verify that mastership has been received:

```
GOLDENGATE% multitool describe –fmt "%n\t%[master]p\n" vob:/vobs/dev
/vobs/dev sanfran_hub@/vobs/dev
```

Transferring Mastership of an Element

When you create a new element, it is mastered by the replica in which you create it. You must perform the following element operations at the element's master replica:

- Changing protections on the element (for ownership-preserving replicas).
- Locking the element with the obsolete option.
- Removing the element.

To transfer mastership of an element to another replica, follow these steps:

1. At the master replica, enter a **chmaster** command:

```
MINUTEMAN% multitool chmaster bangalore tests.txt@@
Changed mastership of file element "tests.txt@@" to "bangalore"
```

2. At the old master replica, export an update packet to the new master replica's site:

```
MINUTEMAN% multitool syncreplica –export –fship bangalore@/vobs/dev
Generating synchronization packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_07-Dec-00.18.15.57_59
78_1
- shipping order file is
/usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_07-Dec-00.18.15.
57_5978_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_07-Dec-00.18.15.57_59
78_1
```

3. At the new master replica, import the packet:

```
RAMOHALLI> multitool syncreplica –import –receive
Applied sync. packet C:\Program
Files\Rational\ClearCase\var\shipping\ms_ship\incoming\sync_boston_hub_07-
Dec-00.18.15.57_5978_1 to VOB \\ramohalli\vobs\dev.vbs
```

4. At the new master replica, verify that mastership has been received:

```
RAMOHALLI> multitool describe –fmt "%n\t%[master]p\n" tests.txt@@
tests.txt@@     bangalore@\dev
```

Transferring Mastership of a Branch

This section describes how to change mastership of a branch using the **chmaster** command. For information about enabling use of the **reqmaster** command, see Chapter 9, *Implementing Requests for Mastership*.

Transferring Branch Mastership

To transfer mastership of a branch to another replica:

1. At the master replica, enter a **chmaster** command:

```
MINUTEMAN% multitool chmaster –c "bugfix at bangalore" bangalore Makefile@@/main
Changed mastership of branch "Makefile@@/main" to "bangalore"
```

2. At the old master replica, export an update packet to the new master replica's site:

```
MINUTEMAN% multitool syncreplica –export –fship bangalore@/vobs/dev
Generating synchronization packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_10-Dec-00.18.15.57_30
56_1
- shipping order file is
/usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_10-Dec-00.18.15.
57_3056_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_10-Dec-00.18.15.57_30
56_1
```

3. At the new master replica, import the packet:

```
RAMOHALLI> multitool syncreplica –import –receive
Applied sync. packet C:\Program Files\Rational\ClearCase\var\shipping
\ms_ship\incoming\sync_boston_hub_10-Dec-00.18.15.57_3056_1 to VOB
\\ramohalli\vobs\dev.vbs
```

4. At the new master replica, verify that mastership has been received:

```
RAMOHALLI> multitool describe –fmt "%n\t%[master]p\n" Makefile@@\main
Makefile@@\main      bangalore@\dev
```

Removing Explicit Mastership of a Branch

As described in *Default and Explicit Branch Mastership* on page 13, a branch can have default or explicit mastership. After you follow the steps in *Transferring Branch Mastership* on page 132, the branch has explicit mastership. When you transfer mastership of a branch type to another replica, mastership is transferred for all branches with default mastership, but not for branches with explicit mastership.

To return mastership of a branch to the replica that masters the branch type:

1. At the replica that masters the branch, enter a **chmaster –default** command:

```
RAMOHALLI> multitool chmaster –default Makefile@@\main
Changed mastership of branch "Makefile@@\main" to "default"
```

2. Determine which replica masters the branch type:

```
RAMOHALLI> multitool describe –fmt "%n\t%[master]p\n" brtype:main
main     boston_hub@\dev
```

   If your current replica masters the branch type, stop here. If another replica masters the branch type, continue with Step #3.

3. Export an update packet to the replica that masters the branch type:

```
RAMOHALLI> multitool syncreplica –export –fship boston_hub@\dev
Generating synchronization packet C:\Program
Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\sync_bangalore_11-D
ec-00.18.15.57_9476_1
- shipping order file is
/usr/atria/shipping/ms_ship/outgoing/sh_o_sync_bangalore_11-Dec-00.18.15.5
7_9476_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/sync_bangalore_11-Dec-00.18.15.57_947
6_1
```

4. At the replica that masters the branch type, import the packet:

```
MINUTEMAN% multitool syncreplica –import –receive
Applied sync. packet
/usr/atria/shipping/ms_ship/incoming/sync_bangalore_11-Dec-00.18.15.57_947
6_1 to VOB /net/minuteman/vobstg/dev.vbs
```

5. At the replica that masters the branch type, verify that the branch has default mastership:

```
MINUTEMAN% multitool describe Makefile@@/main
branch "Makefile@@/main"
  created 27-Aug-00.13:41:21 by Gail Smith (gail.user@boston20)
  branch type: main
  master replica: boston_hub@/vobs/dev (defaulted)
```

The other form of the **chmaster –default** command applies to branches that are explicitly mastered by the replica that masters the branch type. To give these branches default mastership, enter a **chmaster –default** command and specify the branch type:

```
MINUTEMAN% multitool chmaster –default brtype:main
Changed mastership of branch type "main" to "default"
```

Transferring Mastership of a Stream

The **chmaster –stream** command transfers mastership of a stream and its associated objects. For example, to transfer mastership of the stream **v2.1.bl5** and its associated objects to the **boston_hub** replica:

multitool chmaster –stream boston_hub@/vobs/dev stream:v2.1.bl5@/vobs/dev

In some cases, you must manually change mastership of branch types or activities associated with a stream. The output of the **chmaster** command includes a list of these objects. The output may also include an instruction to run the **chmaster –stream** command with the **–override** option. This option transfers mastership of objects whose mastership was not transferred during the original invocation of the command.

CAUTION: Do not use **–override** unless the output of **chmaster –stream** indicates that you should do so.

Transferring Mastership of All Objects Mastered by a Replica

Before removing a replica, you must transfer mastership of all objects mastered by that replica. For detailed instructions, see *Deleting a Replica* on page 119.

The following example shows a partially successful **chmaster –all** command and describes how to fix it. In this example, the administrator at replica **bangalore** is transferring mastership to **boston_hub**.

```
RAMOHALLI> multitool chmaster –all –long boston_hub@\dev
Changed mastership of versioned object base \dev\
Changed mastership of directory element \dev\.@@
Changed mastership of directory element \dev\lost+found@@
...
multitool: Error: Branch type "bangalore_main" has branches (with default
mastership) that have outstanding checkouts.
Changed mastership of branch type v1.0_bugfix
...
multitool: Error: Lock on label type "V1.0_BUGFIX" prevents operation "change
master".
Changed mastership of label type BANGALORE_V2.0
...
Changed mastership of replica bangalore
multitool: Warning: Not all objects had mastership changed.
```

These errors prevent the successful completion of this **chmaster** command:

- There are checkouts on the **bangalore_main** branch.
- There is a lock on a label type.

To fix these problems:

1. Find the checkouts and either check in the files or cancel the checkouts:

   ```
   H:\dev> cleartool lscheckout –all
   03-Jun.17:28 jk checkout version "\dev\cmdsyn.c" from
   \main\bangalore_main\83 (unreserved)
   08-Jun.12:45 singh     checkout version "\dev\etc\util\tool.c" from
   \main\bangalore_main\22 (unreserved)
   ...
   ```

   See the **checkin**, **checkout** and **uncheckout** reference pages.

2. Unlock the type object.

   ```
   cleartool unlock lbtype:V1.0_BUGFIX@\dev
   Unlocked label type "V1.0_BUGFIX".
   ```

   Alternatively, enter a **lock –replace –nusers** command and add yourself to the **–nusers** list.

   ```
   cleartool lock –replace –nusers ms_admin lbtype:V1.0_BUGFIX@\dev
   Locked label type "V1.0_BUGFIX".
   ```

3. Reenter the **chmaster** command.

   ```
   RAMOHALLI> multitool chmaster –all –long boston_hub@\dev
   Changed mastership of branch type bangalore_main
   Changed mastership of label type V1.0_BUGFIX
   Changed mastership of all objects.
   ```

Fixing an Accidental Mastership Change

If a mastership change is made in your replica by mistake, follow these steps to undo the change:

1. At your replica, complete the transfer by sending an update packet to the new master replica.
2. At the new master replica, complete these steps:
   a. Import the packet.
   b. Change mastership back to your replica.
   c. Export an update packet to your replica.
3. At your replica, import the packet.

8.7 Working with Type Objects

When you create an attribute type, a hyperlink type, or a label type, you can make the type shared or unshared. By default, the type is unshared, which means that instances of the type can be created only at the replica that masters the type object. If you define the type object to be shared, instances of the type can be created at any replica in the VOB family.

For more information about type objects, see *Type Object Mastership* on page 14.

Creating a Shared Type Object

To create a shared type object, use the **–shared** option with the **mkattype**, **mkhltype**, or **mklbtype** command. For example, to create a shared attribute type:

cleartool mkattype –shared –c "testing status" TESTED

```
Created attribute type "TESTED".
```

Listing Whether a Type Object Is Shared or Unshared

On Windows, the Properties Browser displays the kind of mastership on the **Mastership** tab.

The **describe** command includes the kind of mastership in its output:

cleartool describe attype:TESTED

```
attribute type "TESTED"
created 15-Aug-00.14:23:27 by Susan Goechs (susan.user@minuteman)
master replica: boston_hub@/vobs/dev
instance mastership: shared
...
```

You can also use the **–fmt** option to display the kind of mastership. For example, to list the mastership kind of a single type object:

cleartool describe –fmt "%n\t%[type_mastership]p\n" attype:TESTED

```
TESTED    shared
```

To list the mastership kind of all label types in a VOB replica:

cleartool lstype –fmt "%n\t%[type_mastership]p\n" –kind lbtype

```
BACKSTOP        shared
BANGALORE_BASE  unshared
BUENOSAIRES_BASE        unshared
CHECKEDOUT      shared
LATEST  shared
BOSTON_BASE        unshared
SANFRAN_BASE    unshared
V1.0    unshared
V2.0    unshared
```

Converting an Unshared Type Object to a Shared Type Object

You can convert an unshared attribute type, hyperlink type, or label type to be shared. For example, if a project manager at the San Francisco site creates an unshared attribute type called **TESTED_BY**, but the Bangalore project manager also needs to use this type, you can convert the type to shared so both project managers can create instances of the type.

NOTE: You cannot convert a shared type object to unshared. To restrict instance creation of a type to one site, remove all instances of the type, remove the type, and create a new unshared type.

For information about using the Properties Browser on Windows to convert an unshared type object to a shared type object, see the MultiSite online help:

1. Click **Start > Programs > Rational ClearCase Administration > MultiSite Help**.
2. On the **Contents** tab of the Help Contents Window, click **Administrator Tasks > To change a type to have shared mastership**.

To use the command line to convert an unshared type object to a shared type object:

1. Determine which replica masters the type object:

   cleartool describe attype:TESTED_BY@/vobs/stage
   ```
   attribute type "TESTED_BY"
     created 03-Oct-00.10:29:06 by John Cole (jcole.user@goldengate)
     master replica: sanfran_hub@/vobs/dev
     instance mastership: unshared
   ...
   ```

2. At the master replica, enter a **mk**type –replace –shared** command to replace the definition of the type:

   cleartool mkattype –replace –shared –c "needed at multiple sites" TESTED_BY
   ```
   Replaced definition of attribute type "TESTED_BY".
   ```

3. Export an update packet to the other sites that must use the type:

   multitool syncreplica –export –fship bangalore boston_hub
   ```
   ...
   ```

4. At the receiving sites, import the update packet:

   multitool syncreplica –import –receive
   ```
   ...
   ```

5. At the receiving sites, confirm that the type object is shared:

   **cleartool describe –fmt "%n\t%[type_mastership]p\n" **
   attype:TESTED_BY@/vobs/dev
   ```
   TESTED_BY     shared
   ```

9 Implementing Requests for Mastership

To support development of elements that cannot be merged, you can give developers the ability to request mastership of branches and branch types. This chapter describes how these requests work, the requirements and recommendations for enabling requests, the planning you must do, and the procedure for enabling requests.

Before reading this chapter, read the information on branch mastership in Chapter 1, *Introduction to MultiSite*, and *Branching and Mastership* on page 36.

9.1 Overview of a Request for Mastership

When a developer requests mastership of a branch, the branch's mastership is transferred to the developer's current replica. When a developer requests mastership of a branch type, mastership of the branch type, along with mastership of all the instances of the branch type that have default mastership, is transferred to the developer's current replica.

The procedure for requesting mastership is as follows:

1. A developer makes a request for mastership.
2. The developer's client host determines which replica masters the branch or branch type, and sends a request for mastership to that replica. This request is made directly to the VOB server, not by sending an update packet.
3. Authorization checking occurs at the sibling replica. The checks are different for a branch and a branch type.

For a request for mastership of a branch, authorization checking determines the following:

a. Whether the developer is allowed to request mastership.

b. Whether requests for mastership of the branch are allowed at the replica level, the branch type level, and the branch level.

c. Whether the replica masters the branch. If the replica does not master the branch, the mastership request fails.

The process in Step #2 uses the information available from the client host's current replica. If the sibling replica has transferred mastership of the branch to another replica, but the current replica has not received an update packet with the change, the information at the current replica is not up to date.

d. Whether the branch, its branch type, or VOB is locked. If one or more of these objects are locked, the request fails.

e. Whether there are any checkouts on the branch, except for nonmastered checkouts. A reserved or unreserved checkout on the branch causes the request to fail.

f. Whether the branch is associated with a stream. You cannot request mastership of a branch associated with a stream.

For a request for mastership of a branch type, authorization checking determines the following:

a. Whether the developer is allowed to request mastership.

b. Whether requests for mastership of the branch type are allowed at the replica level and the branch type level. Also, whether requests are allowed for any of the branch type's instances that have default mastership.

c. Whether the replica masters the branch type. If the replica does not master the branch type, the mastership request fails.

The process in Step #2 uses the information available from the client host's current replica. If the sibling replica has transferred mastership of the branch type to another replica, but the current replica has not received an update packet with the change, the information at the current replica is not up to date.

d. Whether any of the following objects are locked: the branch type, the VOB, or any of the branch type's instances that have default mastership. If one or more of these objects are locked, the request fails.

e. Whether there are any checkouts (except for nonmastered checkouts) on any of the branch type's instances that have default mastership.

f. Whether the branch type is associated with a stream. You cannot request mastership of a branch type associated with a stream.

If the request passes the authorization checks, the process continues with Step #4. (If the developer requests mastership of multiple branches or branch types, error messages are printed for the failures and processing continues.)

4. The server process for the sibling replica assigns mastership of the branch or branch type to the developer's current replica.

   The event record for this operation includes the user name of the requesting developer as part of the comment.

   At this point, the sibling replica is the only replica in the VOB family that has information about the mastership change. At all other replicas in the family, including the developer's current replica, the current mastership information shows that the sibling replica masters the branch or branch type. The developer's current replica is updated when the packet created in Step #5 is imported. The other replicas in the family are not updated until they are synchronized with either of the two replicas that has information about the change.

5. The server at the sibling replica starts an export process to create and send an update packet containing the mastership change to the developer's current replica.

   This packet also contains other changes made since the last synchronization export.

6. The mastership request operation completes its processing.

After the update packet is imported successfully at the developer's current replica, the branch or branch type is mastered by the current replica and developers at the site can create new versions on the branch or create new instances of the branch type.

NOTE: A request for mastership does not initiate a **syncreplica –import** command. If the replica's host uses a receipt handler (the recommended procedure), the import begins as soon as the packet arrives. Otherwise, the import occurs at the scheduled import time at the site or when an administrator imports the packet manually.

9.2 Requirements and Recommendations

To enable requests for mastership in one or more replicas, the following conditions must apply:

- The VOB family is at feature level 2 or higher. (All replicas in the VOB family must be at feature level 2 or higher, even if you are not going to enable requests in all of the replicas.) For more information on feature levels, see Chapter 5, *ClearCase Feature Levels*.

- The sites have high-speed connections (LAN, WAN, T1).

  A request for mastership makes RPCs directly to remote servers and fails if the sites are not connected. If a site has a firewall, developers at that site cannot request mastership from replicas at other sites, and developers at other sites cannot request mastership of any branches mastered at a site with a firewall.

- Each replica masters its own replica object. These replicas are called self-mastering.

  If a replica does not master its own replica object, you cannot enable or disable mastership requests at the replica level. For information about reassigning mastership of the replica object, see *Transferring Mastership of a Replica Object* on page 128.

For mastership requests to work efficiently, the following conditions must apply:

- There is no contention for branches or branch types among the sites. That is, only one person at a time requests mastership of a branch or branch type.

  If two or more developers at different sites compete for mastership of objects, mastership will always be in flux. In this situation, the project leaders and MultiSite administrators must determine whether the branch sharing strategy needs to be changed. Using requests for mastership is not a substitute for using good branching and merging practices.

- The sites exchange update packets frequently.

  Each replica needs current information about object mastership. If a replica is not up to date, requests for mastership from that site cannot determine which replica masters the requested object. Also, if replicas exchange packets infrequently, a mastership request may cause the generation of a large update packet, which will take longer to generate and import.)

- Each replica host uses a receipt handler to import packets.

  You can schedule scripts to import packets regularly. However, to import a packet as soon as it arrives at the replica host, you must use a receipt handler. For more information, see the **shipping.conf** (UNIX) or **MultiSite Control Panel** (Windows) reference page.

9.3 Planning Your Implementation

Before enabling requests for mastership, the project managers and administrators at the different sites must make these decisions:

- Which replicas must be enabled to allow requests for mastership. By default, a replica does not allow requests for mastership. You can enable one replica, multiple replicas, or all replicas in a VOB family.
- Which developers are authorized to request mastership. By default, no one is authorized. You can authorize individual developers, everyone in a specific group, everyone in a specific domain, or everyone in your network.
- The branch types and branches (if any) for which mastership requests are always denied. By default, requests are allowed.

  Although you can enable requests for mastership in components, you cannot request mastership of a branch or branch type associated with a stream.

To Hide Request for Mastership Features

If you do not implement requests for mastership at particular sites, you can hide request for mastership features in the ClearCase graphical interface on Windows. The display of these features is controlled by the site-wide setting **rfm_gui_visibility**.

To use the **setsite** command to hide request for mastership features:

cleartool setsite rfm_gui_visibility=FALSE

To use ClearCase Administration Console to hide request for mastership features:

1. Navigate to the **ClearCase Registry** node in ClearCase Administration Console.
2. Click **Action > Properties**.
3. Click **Help** and follow the instructions in the online help.

9.4 Enabling Requests for Mastership

The procedures in this section use the command line. On Windows, you can use the ACL editor and the Properties Browser. For more information, see the MultiSite online help:

1. Click **Start > Programs > Rational ClearCase Administration > MultiSite Help**.
2. On the **Contents** tab of the Help Contents Window, click **Administrator Tasks > Enabling Requests for Mastership > To enable requests for mastership**.

Prerequisites

1. Ensure that the replica is self-mastering. See *Transferring Mastership of a Replica Object* on page 128.
2. Ensure that the feature level of the replicas in the VOB family is the correct value, and that the VOB family's feature level is the correct value. For instructions, see Chapter 5, *ClearCase Feature Levels*.

Adding Developers to the Access Control List

3. At each replica, add the appropriate people to the replica's access control list.

   multitool reqmaster –acl –edit *vob-selector*

   A replica's access control list (ACL) contains a list of users at other sites who are allowed to request mastership of branches and branch types mastered by that replica. To modify this file, you must be VOB owner, **root** (on UNIX), a member of the ClearCase administrators group (on Windows), or have write permissions on the ACL.

   The *vob-selector* specifies a VOB family, and the ACL for your current replica is changed.

   An access control list contains lines of the following form:

   identity-specification access-level,...

   identity-specification is one of the following:

Everyone	Everyone in all domains.
Domain:*domain*	Everyone in the specified *domain*.

Group:*domain*/*group*	Everyone in the specified *group* in *domain*. You can use a slash (/) or backslash (\ ) between *domain* and *group*.
User:*domain*/*username*	A specific user in a particular domain. You can use a slash (/) or backslash (\ ) between *domain* and *username*.

On Windows, *domain* is the name of a Windows domain (for example, **purpledoc**). On UNIX, *domain* is an NIS domain name (for example, **purpledoc.com**). If someone who can request mastership has user names in multiple domains, you must specify all the identities in the ACL.

access-level is one or more of the following:

Read	Allow read access on ACL
Write	Allow write access on ACL
Change	Allow requests for mastership
Full	Allow requests for mastership and read/write access on ACL

Separate multiple access levels with a comma, but do not include spaces between access levels. The identity specification and associated access levels must appear on the same line.

For example, the following ACL specifies that **susan** can modify the ACL, and **jcole** and **kumar** can request mastership:

```
User:purpledoc.com/susan Read,Write
User:purpledoc/susan Read,Write
User:purpledoc.com/jcole Change
User:purpledoc/jcole Change
User:purpledoc.com/kumar Change
User:purpledoc/kumar Change
```

The following ACL gives **msadm** full permissions and allows everyone to request mastership:

```
User:purpledoc.com/msadm Full
User:purpledoc/msadm Full
Everyone Change
```

Deny Requests for Specific Objects

4. (optional) At each replica, deny requests for mastership of specific objects. By default, requests are allowed for all branches and branch types.

multitool reqmaster –deny *branch-pname*	Denies requests for mastership of the specified branch.
multitool reqmaster –deny *branch-type-selector*	Denies requests for mastership of the specified branch type.
multitool reqmaster –deny –instances *branch-type-selector*	Denies requests for mastership of all instances of the specified branch type.

For you to allow or deny mastership requests for a branch or branch type, your current replica must master it. You can allow or deny mastership requests for all instances of a branch type even if your current replica does not master the type.

If the branch type is a global type, its mastership request setting is stored in the administrative VOB and applies to all local copies of the branch type.

Enable Requests at the Replica Level

5. At each replica, enable requests for mastership at the replica level.

multitool reqmaster –enable *vob-selector*

The *vob-selector* specifies a VOB family, and your current replica is enabled for mastership requests. You must enter this command on the VOB server host.

To enable or disable permission at the replica level, you must be the VOB owner, **root** (UNIX), or a member of the ClearCase administrators group (Windows). Also, the replica must master its own replica object.

In an administrative VOB scenario, you enable requests for mastership in the client VOB replicas. You do not have to enable requests in the administrative VOB replica unless it contains elements that are developed serially.

After you enable requests for mastership, inform the appropriate developers about mastership requests and how and when to use them. *Working On a Team* in the *Working in Base ClearCase* part of *Developing Software* describes the procedures developers must use to request mastership.

NOTE: The **reqmaster** command is a **cleartool** subcommand as well as a **multitool** subcommand, so developers who will request mastership do not have to install MultiSite software on their client hosts. On Windows, developers can request mastership from the Find Checkouts window, the Merge Manager, and the Version Tree Browser.

9.5 Customizing Synchronization Updates for Mastership Requests

After a mastership request is processed at the master replica, **sync_export_list** is invoked to export an update packet to the replica at the requester's site. You can customize the export by specifying one or more of the options and arguments that are valid for **sync_export_list**, except for **–replicas**, which is always the replica at the requester's site.

To specify options and arguments for the export:

1. On the VOB server host of the exporting replica, edit the file **/var/adm/atria/config/rfm_shipping.conf** (UNIX) or *ccase-home-dir***var\config\rfm_shipping.conf** (Windows).

2. Add the options and arguments to the following line:

   ```
   RFM_OPTIONAL_ARGUMENTS =
   ```

   For example, to compress update packets:

   ```
   RFM_OPTIONAL_ARGUMENTS = -compress
   ```

   To suppress informational messages, use a specific shipping class (in this example, **reqmaster**) and compress update packets:

   ```
   RFM_OPTIONAL_ARGUMENTS = -quiet 1 -compress -sclass reqmaster
   ```

On UNIX, MultiSite installation creates the file *ccase-home-dir*/**config/services/rfm_shipping.template**. If **/var/adm/atria/config/rfm_shipping.conf** does not exist, the installation creates it by copying the template file. If **/var/adm/atria/config/rfm_shipping.conf** exists, a note is printed in the installation log advising you to compare the existing file to the template and make any necessary changes.

On Windows, MultiSite installation creates the file *ccase-home-dir***config\services\rfm_shipping.template**. If *ccase-home-dir***var\config\rfm_shipping.conf** does not exist, the installation creates it by copying the template file. If *ccase-home-dir***var\config\rfm_shipping.conf** exists, you must compare the existing file to the template and make any necessary changes.

9.6 Displaying Mastership Request Settings

To display the mastership request setting for a replica, branch type, or branch, use the **describe** command or the **Mastership** tab in the Properties Browser (Windows). These settings are also displayed in the Request Mastership dialog box on Windows.

By default, the output from **describe** shows the mastership request setting. You can also use the **–fmt** option and specify **%[reqmaster]p** to display only the mastership request setting. For example:

➤ To display a replica's mastership request setting:

cleartool describe replica:boston_hub@/vobs/doc

```
replica "boston_hub"
  created 15-Aug-00.14:19:03 by Susan Goechs (susan.user@minuteman)
  replica type: unfiltered
  master replica: boston_hub@/vobs/doc
  request for mastership: enabled
  owner: susan
  group: user
  host: "minuteman"
  identities: preserved
  feature level: 2
  connectivity: connected
  Attributes:
    FeatureLevel = 2
```

cleartool describe –fmt "%[reqmaster]p\n" replica:sanfran_hub@/vobs/dev

```
disabled
```

➤ To display a branch type's mastership request setting:

cleartool describe brtype:main@/vobs/doc

```
branch type "main"
created 15-Aug-00.14:19:03 by Susan Goechs (susan.user@minuteman)
"Predefined branch type used to represent the main branch of elements."
master replica: boston_hub@/vobs/doc
request for mastership: allowed for branch type
request for mastership: allowed for all instances
...
```

cleartool describe –fmt "%[reqmaster]p\n" brtype:boston_main@/vobs/dev

```
denied for all instances
```

➤ To display a branch's mastership request setting:

cleartool describe /vobs/doc/admin/setup.doc@@/main

```
branch "/vobs/doc/admin/setup.doc@@/main"
...
  request for mastership: allowed
...
```

cleartool describe –fmt "%[reqmaster]p\n" /vobs/doc/planning/plans.doc@@/main

```
denied
```

9.7 Troubleshooting

This section describes commands you can use to troubleshoot failed mastership requests, and lists error messages and their meanings.

Troubleshooting Commands

To determine which replica masters a branch or branch type:

➤ Use the **cleartool describe** command. For example:

cleartool describe –fmt "%[master]p\n" file1.txt@@\main

```
boston_hub@\dev
```

cleartool describe –fmt "%[master]p\n" brtype:main@/vobs/doc

```
boston_hub@/vobs/doc
```

➤ (Windows) Display properties of the branch or branch type and click the **Mastership** tab.

To determine whether a mastership request will succeed:

➤ Use **reqmaster –list** (see *Status Messages* on page 152 for descriptions of the output):

multitool reqmaster –nc –list file1.txt@@/main

```
multitool: Error: The following errors will be encountered
multitool: Error: file1.txt@@/main
Request Mastership remote "reqmaster" operation (host "taronga") would
fail:
You do not have permission to request mastership from the sibling replica.
```

➤ (Windows) In the Request Mastership dialog box, click **Preview Request for Mastership**.

To list the event history of a branch or branch type and determine who has requested its mastership, use the **lshistory –minor –fmt** command:

cleartool lshistory –min –fmt "%n\t%o\n%c" file.fm@@/main

```
file.fm@@/main chmaster
Reqmaster changed master replica from "boston_hub" to "buenosaires".
Requester: user "PURPLEDOC\fangio" in domain "PURPLEDOC" on host "mardelplata"
file.fm@@/main chmaster
Reqmaster changed master replica from "tokyo" to "boston_hub".
Requester: user "PURPLEDOC\susan" in domain "PURPLEDOC" on host "minuteman"
file.fm@@/main chmaster
Reqmaster changed master replica from "bangalore" to "tokyo".
Requester: user "PURPLEDOC\masako" in domain "PURPLEDOC" on host "shinjuku"
file.fm@@/main chmaster
Reqmaster changed master replica from "sanfran_hub" to "bangalore".
Requester: user "PURPLEDOC\kumar" in domain "PURPLEDOC" on host "ramohalli"
...
```

cleartool lshistory –min –fmt "%n\t%o\n%c" brtype:main@/vobs/doc

Status Messages

Table 12 describes error messages you may see when you enable or disable requests at the replica level, work with the ACL, and allow or deny requests at the branch type or branch level. Table 13 describes error messages associated with mastership requests.

Errors that occur during the mastership request process, including errors that occur during the synchronization export, are written to the **msadm** log file. To view it, use the **cleartool getlog** command or the ClearCase Administration Console (Windows).

Table 12 Error Messages from Mastership Request Management Operations (Part 1 of 2)

Message	Meaning of message and action to take
`Could not check Request for Mastership permissions.`	The process that checks the ACL could not determine whether you have read or write permissions on the ACL. Check the **msadm** and **albd** log files on the client and server hosts and try the command again later.
`Could not edit Request Mastership ACL.`	You do not have permission to edit the ACL. To edit the ACL, you must be VOB owner, **root** (UNIX), a member of the ClearCase administrators group (Windows), or have write permission on the ACL.
`Could not get Request Mastership ACL.`	Your client computer could not retrieve the ACL from the VOB server host. There may be a network connection problem. Check the **msadm** and **albd** log files on the client and server hosts and try the command again later.
`Could not resolve object '`*object-identifier*`'.`	The command could not find the object. Check the spelling of the object selector. In a dynamic view context, mount the VOB and try the command again.
`Object must be a branch or branch type.`	Specify a branch or branch type. Examples of branch specifications: **/vobs/dev/acc.c@@/main** (UNIX) **\doc\stage.pl@@\main\debug** (Windows) Examples of branch type specifications: **brtype:main** **brtype:boston_main@/vobs/dev** (UNIX) **brtype:v1.0_bugfix@\tests** (Windows)
`Request for mastership ACL operations on multiple replicas are not allowed.`	Specify only one VOB selector.
`The specified selector must be a VOB selector.`	Specify a VOB selector. For example: **vob:/vobs/dev** (UNIX) **vob:\tests** (Windows)
`Request for mastership ACL operations require a VOB-selector argument.`	

Table 12 Error Messages from Mastership Request Management Operations (Part 2 of 2)

Message	Meaning of message and action to take
`The VOB family feature level is too low to enable requests for mastership.`	The VOB family feature level is less than 2. If all replicas in the VOB family are at feature level 2 or greater, raise the family feature level. If any replica in the VOB family has a feature level less than 2, ask the administrator of that replica to upgrade to a newer version of Rational ClearCase (if necessary), raise the feature level of the replica, and send an update packet to the sibling replicas. Raise the family feature level.
`This replica does not master its replica object.`	A replica must be self-mastering for you to enable requests for mastership in that replica. See *Transferring Mastership of a Replica Object* on page 128.
`This replica does not master the branch.`	For you to allow or deny mastership requests for a branch, your current replica must master the branch. Determine which replica masters the branch and ask the administrator of the replica to change mastership of the branch to your replica.
`This replica does not master the branch type.`	For you to allow or deny mastership requests for a branch type, your current replica must master the branch type. Determine which replica masters the branch type and ask the administrator of the replica to change mastership of the branch type to your replica.
`You cannot specify -instances with the -enable option.`	To enable requests at the replica level, use the **–enable** option and specify a VOB selector. To deny or allow requests for all instances of a branch type, use the **–deny** or **–allow** option with the **–instances** option and specify a branch type selector.

Table 13 Error Messages from Mastership Requests (Part 1 of 2)

Message	Meaning of message and action to take
`An error at the sibling replica prevented the request for mastership.`	The error cannot be specified. Try the request again later. If the request continues to fail, ask the administrator of the master replica to check the ClearCase and MultiSite log files.
`At least one checkout prevents the request.`	There is a blocking checkout on the branch being requested or on an instance of the branch type being requested. Try the request again later. If the request continues to fail, ask the user at the sibling replica to check in the element.
`Could not resolve object '`*object-identifier*`'.`	The command could not find the object. Check the spelling of the object selector.
`Locks at the sibling replica prevented the request for mastership.`	A request for mastership fails if the branch or branch type is locked at the master replica. Ask the administrator of the master replica to unlock the branch or branch type.
`Requests are denied for all objects mastered by the sibling replica.`	Mastership requests are not enabled for the replica on host *hostname*. Ask the administrator of the master replica of the branch or branch type to enable mastership requests at the replica level.
`Requests are denied for all objects of the given type.`	Mastership requests are denied for all instances of the branch type. Ask the administrator of the master replica of the branch to use **reqmaster –allow –instances** or the Properties Browser (Windows) to allow requests for all instances.
`Requests are denied for the object.`	Mastership requests are denied for the branch or branch type. Ask the administrator of the master replica to use **reqmaster –allow** or the Properties Browser (Windows) to allow requests for the branch or branch type.
`Requests for mastership can be made only for branches and branch types.`	The user must specify a branch or branch type in the **reqmaster** command. Examples of branch specifications: **/vobs/dev/acc.c@@/main** (UNIX) **\doc\stage.pl@@\main\debug** (Windows) Examples of branch type specifications: **brtype:main** **brtype:boston_main@/vobs/dev** (UNIX) **brtype:v1.0_bugfix@\tests** (Windows)
`The object is not a branch or a branch type.`	

Table 13 Error Messages from Mastership Requests (Part 2 of 2)

Message	Meaning of message and action to take
`The object is already mastered by replica '`*replica-selector*`'.`	The user's current replica already masters the requested object.
`The object was not found at the sibling replica. This may indicate that the replicas are not in sync.`	The user's current replica has more up-to-date information than other replicas in the VOB family. Ask the administrator of the current replica to do both of the following things: ➤ Verify that no update packets are waiting to be imported at other replicas in the VOB family. ➤ Determine whether update packets must be sent more frequently. (Frequent exchange of packets means that replicas have up-to-date information about the state of other replicas.)
`The sibling replica does not master the object.`	The user's current replica has out-of-date information about the mastership of the object. Ask the administrator of the current replica to do both of the following things: ➤ Verify that no update packets are waiting to be imported at your current replica or the sibling replica. ➤ Send update packets more frequently. (Frequent exchange of packets means that replicas have up-to-date information about the state of other replicas.)
`You do not have permission to request mastership from the sibling replica.`	The user requesting mastership is not included on the replica's access control list. Ask the administrator of the sibling replica to use **reqmaster –acl –get** to display the access control list and check the following things: ➤ Spelling of user and domain names ➤ All variants of the domain name are included ➤ User's access level

9.8 Serial Development Scenario

This section describes an example of serial development using requests for mastership.

Planning the Implementation

The company PurpleDoc develops documentation at three sites. There are two VOB families:

- **/vobs/doc** contains binary files. This VOB has three replicas: **boston_hub**, **tokyo**, and **sanfran_hub**.

  The writers working in **/vobs/doc** use serial development because the files are in binary format. However, a team of writers at the Boston site needs control of a certain set of files at all times.

- **/vobs/html** contains html files and scripts. This VOB has three replicas: **boston_hub**, **tokyo**, and **sanfran_hub**.

  The writers working on HTML files in **/vobs/html** use site-specific branch types: **boston_main**, **tokyo_main**, and **sanfran_main**. Writers at a particular site cannot use branch types mastered by the other sites.

  The tool developers working on scripts use the **main** branch. Because the scripts can be merged, the developers can use nonmastered checkouts to do their work.

Setting Up Access Controls

The administrators and project managers at the Boston, San Francisco, and Tokyo sites make the following decisions:

- Writers are allowed to request mastership of all branches in **/vobs/doc**, except for the branches **plans.doc@@/main**, **schedule.doc@@/main**, and **roadmap.doc@@/main**.

- Writers are not allowed to request mastership of any branches of type **boston_main**, **tokyo_main**, or **sanfran_main** in **/vobs/html**.

- Tool developers are allowed to request mastership of all branches of type **main** in **/vobs/html**.

Each administrator completes the following steps on the replica's VOB server host. (This example takes place at the Boston site.)

1. Add writers at other sites to the ACL for **/vobs/doc**.

   a. Place the following lines in the file **/tmp/doc_acl**:

   ```
   # Replica boston_hub@/vobs/doc
   # Request for Mastership ACL:
   User:boston.purpledoc.com/msadm Full
   User:tokyo.purpledoc.com/masako Change
   User:tokyo.purpledoc.com/sato Change
   User:tokyo.purpledoc.com/ito Change
   User:sf.purpledoc/jcole Change
   User:sf.purpledoc/marni Change
   User:sf.purpledoc/david Change
   ```

   b. Use the file to set the replica's ACL:

   multitool reqmaster –acl –set /tmp/doc_acl vob:/vobs/doc

2. Add tool developers at other sites to the ACL for **/vobs/html**.

   a. Place the following lines in the file **/tmp/html_acl**:

   ```
   # Replica boston_hub@/vobs/html
   # Request for Mastership ACL:
   User:boston.purpledoc.com/ccadmin Full
   User:tokyo.purpledoc.com/masako Change
   User:sf.purpledoc/david Change
   ```

   b. Use the file to set the replica's ACL:

   multitool reqmaster –acl –set /tmp/html_acl vob:/vobs/html

   NOTE: After you set the ACL, you can delete the temporary ACL files you created.

3. Deny mastership requests for specific branches and branch types:

   multitool reqmaster –deny /vobs/doc/planning/plans.doc@@/main \ /vobs/doc/planning/schedule.doc@@/main /vobs/doc/planning/roadmap.doc@@/main

   multitool reqmaster –deny –instances brtype:boston_main@/vobs/html

   multitool reqmaster –deny brtype:boston_main@/vobs/html

4. Enable requests for mastership at the replica level.

   multitool reqmaster –enable vob:/vobs/doc vob:/vobs/html

Writing Config Specs

In this scenario, the writers use the config specs listed below. Each location has rules for creating site-specific branches in **/vobs/html** and selecting the latest version on that branch. The **/main/LATEST rule** is used in all the config specs for development in **/vobs/doc** and all other VOBs.

Boston

```
element * CHECKEDOUT
element /vobs/html/scripts/... /main/LATEST
element /vobs/html/files/... /main/boston_main/LATEST
element /vobs/html/files/... /main/LATEST -mkbranch boston_main
element * /main/LATEST
```

San Francisco

```
element * CHECKEDOUT
element /vobs/html/scripts/... /main/LATEST
element /vobs/html/files/... /main/sanfran_main/LATEST
element /vobs/html/files/... /main/LATEST -mkbranch sanfran_main
element * /main/LATEST
```

Tokyo

```
element * CHECKEDOUT
element /vobs/html/scripts/... /main/LATEST
element /vobs/html/files/... /main/tokyo_main/LATEST
element /vobs/html/files/... /main/LATEST -mkbranch tokyo_main
element * /main/LATEST
```

Requesting Mastership

The following sections describe how the writers use mastership requests to do their work.

Serial Development of a File That Cannot Be Merged

1. Masako, in Tokyo, tries to check out the file **\doc\ref\update.fm**, but the checkout fails because the Tokyo replica doesn't master the **main** branch:

   cleartool checkout –c "new command options" \doc\ref\update.fm
   ```
   cleartool: Error: Unable to perform operation "checkout" in replica
   "tokyo" of VOB "\doc".
   cleartool: Error: Master replica of branch "\main" is "boston_hub".
   cleartool: Error: Unable to check out "\doc\ref\update.fm".
   ```

2. She requests mastership of branch **\doc\ref\update.fm@@\main**:

   cleartool reqmaster –c "Tokyo needs mastership" \doc\ref\update.fm@@\main

3. Periodically, she retries the checkout or displays properties of the branch to determine whether mastership has been received.

   cleartool checkout –c "new command options" \doc\ref\update.fm
   ```
   cleartool: Error: Unable to perform operation "checkout" in replica
   "tokyo" of VOB "\doc".
   cleartool: Error: Master replica of branch "\main" is "boston_hub".
   cleartool: Error: Unable to check out "\doc\ref\update.fm".
   ```

   After mastership is received at her replica, the **describe** command shows that her replica masters the branch and her checkout succeeds:

   cleartool describe –fmt "%[master]p\n" \doc\ref\update.fm@@\main
   ```
   tokyo@\doc
   ```

   cleartool checkout –c "new command options" \doc\ref\update.fm
   ```
   Checked out "\doc\ref\update.fm" from version "\main\30".
   ```

Serial Development of a File That Can Be Merged

1. John, in San Francisco, needs to change an HTML script. He can't check out the file using a reserved checkout because the branch is mastered by the Boston replica:

   cleartool checkout –c "option to suppress status msgs" /vobs/html/scripts/conv_fm.pl
   ```
   cleartool: Error: Unable to perform operation "checkout" in replica
   "sanfran_hub" of VOB "/vobs/html".
   cleartool: Error: Master replica of branch "/main" is "boston_hub".
   cleartool: Error: Unable to check out "/vobs/html/scripts/conv_fm.pl".
   ```

2. He requests mastership of the branch:

   cleartool reqmaster –c "SF: add new option" /vobs/html/scripts/conv_fm.pl@@/main

3. He checks out the file with the **–unreserved** and **–nmaster** options and proceeds to edit the file:

   **cleartool checkout –c "option to suppress status msgs" –unreserved **
   –nmaster /vobs/html/scripts/conv_fm.pl
   ```
   Checked out "/vobs/html/scripts/conf_fm.pl" from version "/main/15".
   ```

4. Until mastership is received at the San Francisco replica, he cannot check in his changes:

   cleartool checkin –nc conv_fm.pl
   ```
   cleartool: Error: Unable to perform operation "checkin" in replica
   "sanfran_hub" of VOB "/vobs/html".
   cleartool: Error: Master replica of branch "/main" is "boston_hub".
   cleartool: Error: Unable to check in "conv_fm.pl".
   ```

5. When mastership is received at the San Francisco replica, he attempts to check in the file, but finds that he must perform a merge:

   cleartool checkin –nc conv_fm.pl
   ```
   cleartool: Error: The most recent version on branch "/main" is not the
   predecessor of this version.
   cleartool: Error: Unable to check in "conv_fm.pl".
   ```

6. He performs the merge, and checks in the file:

cleartool merge –to conv_fm.pl –c "merging from LATEST" –version /main/LATEST

```
*******************************
<<< file 1: /vobs/html/conv_fm.pl@@/main/15
>>> file 2: /vobs/html/conv_fm.pl@@/main/16
>>> file 3: conv_fm.pl
*******************************
. . .
Moved contributor "conv_fm.pl" to "conv_fm.pl.contrib".
Output of merge is in "conv_fm.pl".
Recorded merge of "conv_fm.pl".
```

cleartool checkin –nc conv_fm.pl

```
Checked in "conv_fm.pl" version "/main/17".
```

10 Troubleshooting MultiSite Operations

This chapter describes common situations in which running a MultiSite command produces an unexpected result, sometimes accompanied by a warning or error message. The situations fall into these categories:

- **Expected conditions** occur because certain inconsistent changes at different replicas cannot be avoided. In many cases, a MultiSite operation resolves the inconsistency, so you need not take any action.
- **Recoverable errors** are user errors, hardware glitches, and other problems that you resolve by performing a recovery procedure.
- **Serious errors** are problems that may require assistance from Rational Technical Support.

The organization of the descriptions follows the general MultiSite data flow: from replica creation through the phases of replica synchronization—export, transport, and import. This chapter also describes replica restoration and replacement.

For information about changing mastership, see Chapter 8, *Managing Mastership*. For information about mastership request errors, see Chapter 9, *Implementing Requests for Mastership*.

10.1 Troubleshooting Tips

Use the following files and commands to help diagnose MultiSite problems:

- Log files. To view log files, use the **cleartool getlog** command or the ClearCase Administration Console (Windows).

  - MultiSite log files

Export/import problems	Files in directory **/var/adm/atria/log/sync_logs** (UNIX) or *ccase-home-dir***var****log** (Windows)
Transport problems	**shipping**
Mastership request problems	**msadm**
Other errors	Command window Event Viewer (Windows)

  - ClearCase log files. If ClearCase problems affect MultiSite operation (for example, a MultiSite operation fails when the ClearCase **db_server** cannot process the VOB database), useful information appears in these log files.

- Make sure you install the latest ClearCase and MultiSite patches.

- Most MultiSite commands do not require a view context or a mounted VOB replica. If a command such as **syncreplica –import** fails, you can produce better diagnostics by following the steps below.

  On UNIX:

  a. Set a dynamic view or change to a directory within a snapshot view.

  b. Mount the VOB replica (dynamic view) or load a single file in the VOB (snapshot view).

  c. Change into a directory in the replica. If you used a snapshot view, this must be the directory containing the file you loaded.

  d. Enter the command again.

  On Windows:

  a. Change to a directory within a snapshot view or to a view drive.

  b. Mount the VOB replica (dynamic view) or load a single file in the VOB (snapshot view).

c. Change into a directory below the root directory. If you used a snapshot view, this must be the directory containing the file you loaded.

d. Enter the command again.

➤ The commands listed below provide valuable information, especially if you are sending data to Rational Technical Support:

multitool –version
multitool lsreplica
multitool lsepoch
uname –a (UNIX)
cleartool –version

On Windows, look for applicable messages in the Event Viewer's application log and system log, and in the ClearCase MVFS log files (**c:\mvfslogs**).

10.2 Replica-Creation Problems

Problems with replica creation can occur during the export phase or the import phase.

Export Phase

If the **mkreplica –export** command finds that a replica with the specified name exists in the VOB family (`Replica` *replica-name* `already exists`), select another name for the new replica, and reenter the **mkreplica –export** command.

If **mkreplica –export –fship** fails while it is transporting the packet, it does not remove the new replica's replica object at the creating site. To complete the replica creation, use **shipping_server** to transfer the replica-creation packet.

Import Phase

A recoverable error occurs if the **mkreplica –import** command detects a conflict at the ClearCase registry level—an entry exists in the VOB *object registry* or in the *tags registry*:

`Replica` *replica-name* `already exists`

Conflict in VOB Object Registry

A conflict in the registry can occur if a **mkreplica –import** commands fails and removes the VOB storage directory but not the registry entry. Verify that **cleartool lsvob** does not report any VOB storage directory at the location you specified with the **–vob** option. In this case, the VOB *object registry* contains an entry with no corresponding VOB-tag. For example:

cleartool lsvob –storage /net/goldengate/vobstg/dev.vbs

```
cleartool: Error: Unable to access "/net/goldengate/vobstg/dev.vbs": No such
file or directory.
cleartool: Error: Versioned object base not found:
"/net/goldengate/vobstg/dev.vbs".
cleartool: Error: No vob tags found for vob "/net/goldengate/vobstg/dev.vbs".
```

Restore the registry to a consistent state by following these steps:

1. In the VOB object registry file, find the incorrect entry for the VOB storage directory pathname you specified. This file is located on the network's registry server host in **/var/adm/atria/rgy/vob_object** on UNIX or *ccase-home-dir***var\rgy\vob_object** on Windows.

2. Using the UUID in this entry, enter a **cleartool unregister –vob –uuid** command to remove the incorrect entry.

   CAUTION: Do not edit the information in the registry file directly.

3. With the registry restored to a consistent state, reenter the **mkreplica –import** command.

4. After the **mkreplica** command succeeds, delete the replica-creation packet from disk storage (if appropriate).

Conflict in VOB-Tag Registry

mkreplica –import may be able to create and register the VOB storage directory, but may find that the specified VOB-tag is already in use. In this case, create another VOB-tag for the new VOB storage directory with a **cleartool mktag** command or with the ClearCase Administration Console (available on Windows).

You do not have to reenter the **mkreplica –import** command in this case. You can delete the replica-creation packet from disk storage (if appropriate).

10.3 Synchronization Export Problems

This section describes problems that can occur during the export phase of synchronization.

To list the exports from your current replica to a sibling replica, use the following command:

cleartool lshistory replica:*sibling-replica-name@vob-selector*

For example, to list exports from your current replica in VOB family **/vobs/dev** to the replica **sanfran_hub**:

cleartool lshistory replica:sanfran_hub@/vobs/dev

```
12-Jul.16:13 root export sync from replica "boston_hub" to replica
"sanfran_hub"
  "Exported synchronization information for replica "sanfran_hub".
  Row at export was: boston_hub=149 sanfran_hub=115"
29-Jun.16:19 smg change epoch of replica "sanfran_hub"
  "Changed epoch row for replica
  Old row was: boston_hub=152 sanfran_hub=115
  New row is: boston_hub=149 sanfran_hub=115
  epoch row set by special connected epoch tool."
29-Jun.10:12 smg export sync from replica "boston_hub" to replica
"sanfran_hub"
  "Exported synchronization information for replica "sanfran_hub".
  Row at export was: boston_hub=149 sanfran_hub=115"
...
```

Cannot Find Oplog

syncreplica –export can fail with the following warning message:

```
Can not find oplog from replica replica-name with id oplog-ID
Gap in oplog entries may indicate missing oplog entries
```

(For more information on oplog entries, see *VOB Operations and the Oplog* on page 24 and *Scrubbing Parameters for VOB Replicas* on page 47.)

This error occurs when the sending replica's epoch number matrix does not match its set of oplog entries. For example:

- Before sending an update from **sydney** to **buenosaires**, **syncreplica** checks the epoch number matrix for **sydney**. It determines that the last **sydney** operation sent to **buenosaires** was 3620.
- **syncreplica** finds that oplog scrubbing in the **sydney** database has removed some of the operations that follow 3620. The earliest **sydney** operation remaining in the oplog is 5755.

This discrepancy may be an expected condition. For example, when a VOB family changes its update topology, hosts that have not communicated with each other in the past start exchanging update packets. Synchronizing two replicas (**syncreplica –export** followed by **syncreplica –import**) updates epoch number matrix rows for the sending and receiving replicas, but it does not revise the row for any other replica. If two replicas rarely (or never) send updates to each other directly, the relevant rows in their epoch number matrices are out of date (possibly consisting of all zeros). This is not a problem, as long as the replicas receive operations indirectly, for example, through a hub replica.

In this case, you must inform **sydney** about the true state of **buenosaires**, information that it has not received through the standard synchronization-update mechanism. This information enables **sydney** to determine which oplog entries must be sent to **buenosaires**.

If the sites have an IP connection, use the procedure in *Sites Have IP Connection*. If the sites do not have an IP connection, use the procedure in *Sites Do Not Have IP Connection*.

Sites Have IP Connection

At **sydney**, use the **chepoch –actual** command to contact **buenosaires**, retrieve its actual state, and reset the epoch row for **buenosaires**.

multitool chepoch –actual replica:buenosaires@/vobs/tests

Sites Do Not Have IP Connection

Proceed as follows:

1. At **buenosaires**, use the **lshistory** command to determine when the last update packet was processed successfully.

   cleartool lshistory replica:buenosaires@/vobs/tests

   ```
   01-Sep.01:00 garyf  import sync from replica "sydney" to replica
   "buenosaires"
   "Imported synchronization information from replica "sydney".
   Row at import was: buenosaires=8 sydney=3 boston_hub=0"
   01-Aug.07:05 garyf  import sync from replica "boston_hub" to replica
   "buenosaires"
   "Imported synchronization information from replica "boston_hub".
   Row at import was: buenosaires=2 boston_hub=0"
   01-Jul.15:55 garyf  create replica "buenosaires"
   ```

2. At **sydney**, use this time in a **recoverpacket** command to reset the epoch row for **buenosaires**. Assume that the **sydney** site is thirteen hours ahead of the **buenosaires** site.

   multitool recoverpacket –since 01-Sep.14:00 buenosaires

   If this command succeeds, proceed to Step #3.

   If this command fails:

   a. At **buenosaires** (destination site), run **lsepoch** to determine the actual state of **buenosaires**:

      multitool lsepoch buenosaires@/vobs/tests

   b. Send the **lsepoch** command output back to the sending site, where the administrator of **sydney** uses this data in a **chepoch** command to inform **sydney** about the actual state of **buenosaires**.

      cd /vobs/dev
      multitool chepoch buenosaires
      `Enter specifications for epochs to change in row "buenosaires" (one per line)`
      <output of lsepoch command>
      .

3. At **sydney**, enter the original **syncreplica –export** command.

   - If the command fails, **buenosaires** is in jeopardy. Have other replicas in the VOB family perform Step #1 through Step #3, taking the role of **sydney** to exchange update packets with **buenosaires**. The hope is that some other replica has not yet scrubbed its copies of the missing oplog entries. If no other replica has the missing oplog entries, you must create a new replica. See *Replacing an Existing Replica* on page 192.

   - If the command succeeds and the packet is imported successfully at **buenosaires**, **buenosaires** is up to date.

NOTE: Have all sites review their oplog scrubbing procedures. You may have to change the **oplog –keep** settings in one or more **vob_scrubber_params** files. See *Scrubbing Parameters for VOB Replicas* on page 47.

Oplog Gap Detected During Creation of Update Packet

syncreplica –export can fail with the following warning message:

```
Gap in oplog detected for replica replica-name.
Wanted oplog id: oplog-ID.  Got oplog id: oplog-ID.
```

This error message can indicate a serious error, involving an unrecoverable data loss. If the procedures described in *Cannot Find Oplog* on page 168 do not work, contact Rational Technical Support.

Export Failure During Version Construction

An export operation can fail with a message like the following:

```
multitool: Error: Type manager "z_text_file_delta" failed construct_version
operation.
multitool: Error: Could not get statistics of the version data file for this
operation.
multitool: Error: Synchronization update terminated prematurely due to error
-- aborting.
```

This situation can occur when an export operation tries to access an element that is being modified by a user. In this case, retry the export.

Packets Accumulate in Outgoing Storage Bay

Problems with packet delivery are recoverable errors. In many cases, the MultiSite automatic-retry capability recovers from errors.

A replica-creation or update packet submitted to the *store-and-forward* facility for transport to one or more other hosts is accompanied by a *shipping order* file. (A logical packet can include multiple physical packets, each with its own shipping order.) The shipping order typically has an expiration time, determined by one of the following:

- A date-time specified with the **–pexpire** option in the **syncreplica** or **mkreplica** command that generated the packet (or the **mkorder** command that submits an arbitrary file to the store-and-forward facility)
- On UNIX, the **EXPIRATION** value in the *store-and-forward* configuration file (**shipping.conf**) on the sending host
- On Windows, the Packet Expiration value specified in the **MultiSite Control Panel** on the sending host

Any number of delivery attempts may take place before the shipping order expires.

Replica Cannot Update Itself

You can receive the following message during export if you specify the sending replica as a destination:

```
A replica cannot update itself
```

If the sending replica is the only replica you specified, the **syncreplica –export** command fails. If you specified other replicas, this message is printed as a warning, and the **syncreplica –export** command continues its processing.

10.4 Transport Problems

This section describes problems that can occur during the transport phase of synchronization.

Error Messages

The messages in Table 14 are generated by the **mkorder**, **mkreplica**, **shipping_server**, and **syncreplica** commands.

Table 14 Shipping Error Messages (Part 1 of 2)

Error message	Meaning
`cannot find a storage bay for class` *class-name*`: no such bay specified`	No storage bay is assigned to storage class *class-name* in the **shipping.conf** file or the **MultiSite Control Panel**.
`cannot find a storage bay for class` *class-name*`: all applicable bays are either inaccessible or do not contain` *byte-count* `free bytes`	Lack of permission or lack of free disk space prevents use of storage bays for class *class-name*.
`cannot process potential order file` *shipping-order-pname*`: user` *username* `(UID` *uid*`) is not the owner (UNIX)`	**shipping_server** is not running as **root**, and *username* does not own the shipping order file.
`cyclic delivery route detected to host` *hostname* `(via` *next-hop-hostname*`) for order` *shipping-order-pname*	Sending the file to the *next-hop-hostname* specified in a **ROUTE** entry in the **shipping.conf** file or in the **Routing Information** section in the **MultiSite Control Panel** yields a circular delivery route.
`file` *file-pname* `does not contain a valid shipping order`	**shipping_server** attempted to process a file that is not a shipping order.
`for security reasons, shipping order` *shipping-order-pname* `cannot be processed: data file` *file-pname* `must be in the same directory as the shipping order`	A shipping order and its associated packet file must be in the same directory. This security feature prevents transmission of arbitrary files.

Table 14 Shipping Error Messages (Part 2 of 2)

`giving up trying to return order` *shipping-order-pname* `to host` *hostname* `(original data file was` *file-pname*`)`	**shipping_server** cannot return a packet or other file to its original sending host (for example, because its shipping order expired) and has deleted the shipping order and data file.
`ignoring shipping bay` *storage-bay-pname*`:` *reason*	The storage bay directory specified in the **shipping.conf** file or **MultiSite Control Panel** is inaccessible, doesn't exist, and so on.
`shipping order` *shipping-order-pname* `not found (perhaps previously sent?)`	During receipt handler processing, the **shipping_server** cannot find the shipping order of a packet that is to be forwarded to another host. A **shipping_server –poll** invocation may have sent the packet already. (If the packet is to be applied to replicas on the host, the imports occur before the packet is forwarded. This leaves a window of opportunity for a scheduled polling operation to send the packet.)

Invalid Destination

The local host's **hosts** file, **hosts** NIS map, or Domain Name Service must list one of the following hosts:

- Destination host
- Next-hop host corresponding to the destination host (on UNIX, defined in a **ROUTE** entry in the host's **shipping.conf** file; on Windows, defined in the **Routing Information** section in the host's **MultiSite Control Panel**.)

NOTE: If hosts in your network are known only by their IP addresses, you can use the IP addresses instead of host names.

In the absence of such entries, **shipping_server** fails, because it cannot determine where to deliver the packet. In this case, it writes error messages to its log file.

If the destination host name was misspelled, use the **mkorder** command to create a new shipping order with the correct host name. If a host name is misspelled in a **mkreplica –export** command, the incorrect host name is recorded in the VOB database. Verify the error with **lsreplica –long**, and correct the spelling with **chreplica**.

In other cases, you may have to revise the host's database of remote hosts. The sending host must be able to communicate with the receiving hosts through TCP/IP channels. Use the **rcp** command on the sending host to copy a file to the receiving host. If it fails, you have a setup or networking problem with your host. If the command succeeds, contact Rational Technical Support.

Delivery Fails

Each time **shipping_server** cannot deliver a packet to a valid destination host, it logs error messages:

- (On UNIX) In file **/var/adm/atria/log/shipping_server_log** and writes a message to the terminal device, if there is one.
- (On Windows) In the Windows event viewer. It writes log messages to file *ccase-home-dir***var\log\shipping_server_log**.

If the problem is temporary (remote host is down, network connections are down, and so on), a subsequent invocation of **shipping_server –poll** will transmit the packet successfully. If the problem is not temporary, the shipping order may expire eventually.

Shipping Server Fails to Start or Connection Is Refused

If the **shipping_server** at the receiving site does not start or the connection is refused, check the **albd_server** log on the receiving host for an explanation of the failure.

A syntax error in the **shipping.conf** file on UNIX can cause the connection to be refused. For example, if there is an incorrect e-mail address in the file, the **albd_server** log displays an error like this:

```
Error: shipping_server(9951): Error: syntax error in configuration file (line
60)
```

Shipping Order Expires

If the **shipping_server** finds that a shipping order has expired, it attempts to return the packet to the originating host. Also, it sends a mail message to one or more administrators on the original sending host, and sends another mail message when the packet is returned to the original sending host. On Windows, if e-mail notification is not enabled, **shipping_server** writes a message to the Windows event viewer and records the error in the *ccase-home-dir***var****log****shipping_server_log** file.

Use the **lspacket** command to check the *return bays* on your host. The packet files may have been returned by store-and-forward. If so, try again to deliver the packet:

- Fix the store-and-forward packet-delivery mechanism (for example, by fixing the network connection). Then, use **mkorder** to create a new shipping order for each physical packet file in the return bay.
- If you cannot fix the store-and-forward mechanism, deliver the packet by some other means. For example, copy the packet file to a diskette, and mail the diskette to the remote sites.

If the packet files are not in your host's return bays, they may be in transit. Search for the files immediately, because a packet that cannot be returned to its home host within 14 days is deleted.

10.5 Synchronization Import Problems

This section describes problems that can occur during the import phase of synchronization.

To list the imports at your current replica, use the following command:

cleartool lshistory replica:*current-replica-name@vob-selector*

For example, to list imports at the replica **boston_hub** in VOB family **/vobs/dev**:

cleartool lshistory replica:boston_hub@/vobs/dev

```
25-Jun.11:46   smg       import sync from replica "sanfran_hub" to replica
"boston_hub"
  "Imported synchronization information from replica "sanfran_hub".
  Row at import was: boston_hub=149 sanfran_hub=112"
10-Jun.12:36   smg       import sync from replica "sanfran_hub" to replica
"boston_hub"
  "Imported synchronization information from replica "sanfran_hub".
  Row at import was: boston_hub=136 sanfran_hub=111"
10-Jun.12:01   smg       import sync from replica "sanfran_hub" to replica
"boston_hub"
  "Imported synchronization information from replica "sanfran_hub".
  Row at import was: boston_hub=135 sanfran_hub=63"
```

Packets Accumulate in Incoming Storage Bay

A recoverable error occurs when an update packet is lost and is not applied at your site. These are the symptoms:

- One or more replicas at your site are not being updated on their regular schedules.
- An **lspacket** command shows unprocessed packets accumulating in the storage bay. These packets depend on the missing packet and cannot be processed.

Verify that a packet is missing and determine which operations are needed:

1. Enter a **multitool syncreplica –import –receive** command, which processes all incoming packets in the storage bay in the correct order. If **syncreplica** refuses to process any of them, a packet is missing.
2. Enter a **syncreplica –import** command that specifies the oldest packet in the storage bay:

   multitool syncreplica –import *packet-pathname*

   ```
   Sync. packet packet-pathname was not applied to VOB ...
    - packet depends on changes not yet received
   Packet requires changes up to 872; VOB has only 756 from replica:
   sanfran_hub
   Packet requires changes up to 605; VOB has only 500 from replica:
   bangalore
   ```

In this example, one or more update packets are missing, containing operations 757–872 originally occurring at replica **sanfran_hub** and operations 501-605 from **bangalore**. In general, a packet can contain operations from several replicas; the **syncreplica –import** command fails if operations are missing from *any* replica.

Locate the missing packets. They may be on a magnetic tape that you forgot to process or in packet files that were not processed because your store-and-forward configuration (the **shipping.conf** file on UNIX; the **MultiSite Control Panel** on Windows) specifies the wrong storage bay. If you locate the missing packets:

1. Process the missing packets by naming them in a **syncreplica –import** command. (Multiple packet files are imported in the correct order, regardless of the order of the command-line arguments.)

2. Process all the update packets that have accumulated in the storage bay by entering a single **syncreplica –import –receive** command.

If you cannot locate the missing packets, go to *Recovering from Lost Packets* on page 182.

Packet is Not Applicable to Any Local VOB Replicas

Import can fail with the following message:

```
multitool: Error: Sync. packet pathname is not applicable to any local VOB
replicas
```

This error can occur when a replica has been moved and the hostname property has not been updated with the **chreplica** command. To verify that the hostname property is wrong, enter the following command:

cleartool describe –fmt "%[replica_host]p\n" replica:*importing-replica-name@VOB-tag*

For example:

cleartool describe –fmt "%[replica_host]p\n" replica:newyork@/vobs/tests
```
manhattan
```

If the hostname is incorrect, use the **chreplica** command to change it. At the master replica of the importing replica, enter this command:

multitool chreplica –c "*comment***" –host** *new-host* **replica:***importing-replica-name@VOB-tag*

For example:

multitool chreplica –c "change hostname" –host brooklyn replica:newyork@/vobs/tests
```
Updated replica information for "newyork".
```

Send an update packet to the other replicas in the VOB family.

Read from Input Stream Fails

If a **syncreplica –import** command fails with a message like this one, the packet is corrupted:

```
multitool: Error: Read from input stream failed: No such file or directory
```

Delete the packet and ask the administrator at the sending site to re-create the packet and send it again (see *Recovering from Lost Packets* on page 182). Then import it.

Element Changes During Operation

If a **syncreplica –import** command fails with one of the following messages, restart the import:

```
Element changed during operation
Element changed during checkin
```

The messages report that **multitool** was trying to import an operation for an element while another process (for example, a developer using **cleartool**) was operating on the same element.

If possible, restart the **syncreplica –import** from within a view. If it fails again, you see more information about what element it is failing on, and you can look through output from the **lshistory** command to try to find the conflict.

rmreplica Operation Cannot be Imported

Import of an **rmreplica** operation fails if the importing replica thinks that the removed replica still masters objects. The import fails with an error like the following:

```
multitool: Error: There are still objects mastered by this replica.
multitool: Error: Unable to replay oplog entry 565632: error detected by
ClearCase subsystem.
565632:
12  op= rmreplica
13  replica_oid= 48abc01d.123456a7.b890.06:00:08:c4:73:84 (boston_hub.mstr)
14  oplog_id= 23456
15  op_time= 08/07/00 12:35:46 create_time= 08/07/00 12:35:46
16  event comment= "Destroyed replica "boston_hub".
```

This situation can occur if two VOB replica hosts do not have the same patch level or if a ClearCase upgrade had problems.

You can use the **lsmaster** command to determine which objects are believed to be mastered by the removed replica. In this example, the administrator at importing replica **sanfran_hub** uses the **lsmaster** command to list the objects replica **sanfran_hub** believes to be mastered by replica **boston_hub**:

multitool lsmaster –view admin_view boston_hub@/vobs/dev

```
master replica: boston_hub@/vobs/dev  "label type" V2.0
master replica: boston_hub@/vobs/dev  "label type" V1.1
```

In this example, the administrator at replica **sanfran_hub** uses the **lsmaster** command to contact all replicas in the VOB family and list the objects they believe to be mastered by replica **boston_hub**:

multitool lsmaster –view admin_view –inreplicas –all boston_hub@/vobs/dev

```
In replica "bangalore"
master replica: boston_hub@/vobs/dev "label type" V2.0
In replica "sanfran_hub"
master replica: boston_hub@/vobs/dev  "label type" V2.0
master replica: boston_hub@/vobs/dev  "label type" V1.1
```

To resolve this problem, contact Rational Technical Support.

Replica Incarnation is Old

The following error can occur during packet import:

`multitool: Error: Replica incarnation for "REPLICA_NAME" is old:` *old-timestamp* `should be` *new-timestamp*

The replica incarnation is the last time the replica was restored (with the **restorereplica** command). The incarnation is set to `0` when the replica is created and remains `0` until a restoration occurs.

Each replica keeps a record of the incarnation of each other replica in the VOB family. During packet export, the incarnations of the target replicas are recorded in the packet. The **syncreplica –import** command at the importing replica checks the incarnation in the packet. If the incarnation in the packet is earlier than the importing replica's own record of its incarnation, the packet is not imported.

If the incarnations are different, the exporting replica does not have a record of the importing replica undergoing restoration. This situation may occur for the following reasons:

- The update packet was created before the restoration information arrived at the exporting replica.
- The restoration information was not sent to the exporting replica. For example, consider the following synchronization setup:

  Replicas A and B synchronize every day, Replicas B and C synchronize once a week, and Replicas A and C synchronize once a month.

  Replica A is restored from backup and the administrator runs **restorereplica**. Because Replica A's last synchronization was with Replica B, the administrator optimizes the process to require an update packet only from Replica B. After the packet is received from Replica B, the restoration is complete and Replica A resumes normal synchronization.

  Because neither Replica A nor Replica B synchronized with Replica C during the restoration process, Replica C does not have any information about the restoration, and its record of Replica A's incarnation is not updated.

  The next time Replica C sends an export packet to Replica A, the incarnation in the packet is earlier than Replica A's actual incarnation, and the import fails.

To determine which reason applies to your situation:

1. At the exporting replica, display the incarnation time for the importing replica.

   cleartool dump replica:*name-of-importing-replica@VOB-tag*

   In the output, look for a line beginning with `incarnation=`. This line displays the incarnation time. For example:

   cleartool dump replica:boston_hub@/vobs/dev
   ```
   ...
   incarnation=01-Apr-99.22:40:54UTC
   ...
   ```

2. Compare this value to the value in the import error message.

   - If the values are the same after you adjust for time zone differences, the packet was created before the exporting replica received the restoration information. Delete the packet and follow the instructions in *Recovering from Lost Packets* on page 182.

   - If the values are different, contact Rational Technical Support.

Miscellaneous Problems

Processing of an incoming replica-creation or update packet may fail because of these conditions:

- Disk partition is full.
- Receiving replica is locked.
- ClearCase or MultiSite licensing failure.
- Multiple imports occur simultaneously.

Make sure that multiple **syncreplica –import** commands do not run in the same replica simultaneously. Check the timing of **schedule** tasks, and adjust them if necessary. (An invocation of the **sync_receive** script fails if another sync_receive process is running.)

In such cases, fix the problem and reenter the **syncreplica –import** command.

Recovering from Lost Packets

There are several circumstances in which a replica-creation or update packet is generated but is never applied at one or more of its destinations:

- The packet is stored on media that are destroyed or are not readable at the destination host.
- A packet file is lost when a hard disk fails.
- The packet is intact, but cannot be applied because another packet has been lost. (See *Packets Accumulate in Incoming Storage Bay* on page 176.)

Lost Replica-Creation Packet

To recover a lost replica-creation packet:

1. At the replica where the **mkreplica –export** command was entered, remove the new replica with **rmreplica**.
2. Reenter the **mkreplica** command.

Lost Update Packet

The **syncreplica –export** command assumes successful delivery of the update packet it generates. For example, when replica **boston_hub** sends an update to replica **sanfran_hub**, the **syncreplica** command assumes that the operations originating at **boston_hub** are imported to the **sanfran_hub** replica. For simplicity, this example does not reflect the fact that the update packet can also contain operations that originated at other replicas in the VOB family.

However, if the packet is lost, this assumption is invalid, and **boston_hub** must reset its estimate of the state of replica **sanfran_hub**. After this correction is made, the next update packet sent from **boston_hub** to **sanfran_hub** contains the operations **sanfran_hub** needs.

To reset the epoch row, use one of the methods described here.

Method 1

1. At the sending site, use **sync_export_list –update** or **chepoch –actual** to set the epoch row to match the actual state of the receiving replica. These commands contact the receiving replica and retrieve its epoch row (the receiving replica's record of its own state). The **sync_export_list –update** command sends an update packet after it updates the epoch row in the sending replica. The sending and receiving sites must have an IP connection.

For example, use one of the following commands:

/usr/atria/config/scheduler/tasks/sync_export_list –update –replicas sanfran_hub@/vobs/dev

multitool chepoch –actual sanfran_hub@/vobs/dev

```
Entry for        bangalore changed from:         985 to         950
Entry for       boston_hub changed from:        1400 to        1300
Entry for      sanfran_hub changed from:        2562 to        2000
```

Method 2

1. At the receiving site, use the **lsepoch** command to display the replica's epoch number matrix:

   multitool lsepoch sanfran_hub@/vobs/dev

   ```
   For VOB replica "/vobs/dev":
   Oplog IDs for row "sanfran_hub" (@ goldengate):
    oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=950        (bangalore)
    oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=1300       (boston_hub)
    oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=2000       (sanfran_hub)
   ```

2. Use this output in a **chepoch** command at the sending site:

   multitool chepoch sanfran_hub bangalore=950 boston_hub=1300 sanfran_hub=2000

   ```
   Change oplog ID in row "sanfran_hub", column "bangalore" to 950 [no] yes
   Change oplog ID in row "sanfran_hub", column "boston_hub" to 1300 [no] yes
   Change oplog ID in row "sanfran_hub", column "sanfran_hub" to 2000 [no]
   yes
   Epoch row successfully set.
   ```

Method 3

1. At the sending site, use **lshistory** to determine the epoch numbers when the packet was generated:

   cleartool lshistory –long replica:sanfran_hub

   ```
   30-Jul.14:42:50     Susan Goechs (susan.user@minuteman)
     export sync from replica "boston_hub" to replica "sanfran_hub"
     "Exported synchronization information for replica "sanfran_hub".
     Row at export was: bangalore=950 boston_hub=1300 sanfran_hub=2000"
   23-Jul.17:36:46     Susan Goechs (susan.user@minuteman)
     export sync from replica "boston_hub" to replica "sanfran_hub"
     "Exported synchronization information for replica "sanfran_hub".
     Row at export was: bangalore=900 boston_hub=800 sanfran_hub=1500"
   ...
   ```

2. At the sending site, use this output in a **chepoch** command:

   multitool chepoch sanfran_hub bangalore=950 boston_hub=1300 sanfran_hub=2000
   `Change oplog ID in row "sanfran_hub", column "bangalore" to 950 [no]` **yes**
   `Change oplog ID in row "sanfran_hub", column "boston_hub" to 1300 [no]` **yes**
   `Change oplog ID in row "sanfran_hub", column "sanfran_hub" to 2000 [no]` **yes**
   `Epoch row successfully set.`

Method 4

1. At the site that failed to apply the lost packet, use the **lshistory** command to determine the time of the last successful import of an update packet from the site that sent the lost packet.

   `GOLDENGATE>` **cleartool lshistory replica:sanfran_hub**
   `01-Aug.07:08 jcole  import sync from replica “boston_hub” to replica “sanfran_hub”`
   `“Imported synchronization information from replica “boston_hub”.`
   `Row at import was: sanfran_hub=2000 boston_hub=1300 bangalore=950”`
   `...`

2. At the sending site, use this time in a **recoverpacket** command. **recoverpacket** looks through epoch rows to find an event that occurred prior to the specified time. When it finds a matching row, it resets the epoch row for the receiving site.

   `susan@minuteman%` **multitool recoverpacket –since 01-Aug.01:00 sanfran_hub**

NOTE: With this method, you must adjust the time from the **lshistory** output for time zone differences and the amount of time elapsed between export and import.

If there are no saved epoch rows for the receiving replica that are as old as the time specified, you must use one of the **chepoch** procedures.

Inconsistent Changes to Replica

A recoverable error occurs if **syncreplica –import** detects that an incoming change is inconsistent with another change that has already been applied to the replica.

NOTE: In some cases, an inconsistency is resolved by **syncreplica –import**. For example, a replica receives an update that deletes an element, then receives an update from another replica that creates a new version on a branch of that element. The create-version operation in the second update is discarded because the element no longer exists.

Ownership Preservation

If two replicas are ownership-preserving, the OS-level permissions of their individual elements are synchronized. However, synchronizing the VOB group lists of the replicas is a manual task that you perform using **cleartool protectvob –add_group**.

syncreplica –import generates the following ownership-related error messages:

```
Can't create object with group that is not in the VOB's group list
Can’t change to a group that is not in the VOB’s group list
```

These messages indicate that the sending replica added a group to its VOB group list, created a new element in that group or reassigned an existing element to that group, and sent the ownership change to a replica whose VOB group list has not been updated.

These messages may also indicate that the sending replica and/or receiving replica were created incorrectly as ownership-preserving.

If the replicas are intended to be ownership-preserving, follow these steps to recover from this kind of error:

1. (If necessary) Set a view, change to a directory within the replica, and reenter the **syncreplica –import** command. This produces diagnostics that include pathnames within VOB directories. For example:

   ```
   elem_fstat= ino: 0; type: 2; mode: 0444; uid: 1037; gid: 20
     .
     .
   name_p= "aux_util.c"
   nsdir_ver_oid= ed2549e2.97f411cd.b3c8.08:00:69:06:4d:f6
                  (/vobs/dev/src@@/main/ev2/CHECKEDOUT.572)
   ```

   These lines indicate that the element’s pathname in the sending replica is **/vobs/dev/src/aux_util.c**. Note also that its group-ID (GID) is 20.

2. Use the **cleartool protectvob** command to add the new group to your replica’s VOB group list:

   cleartool protectvob –add_group 20 /vobstg/dev.vbs

3. Reenter the **syncreplica –import** command.

NOTE: If the administrators at the sites of ownership-preserving replicas have not informed one another of changes in the shared user/group namespace, you may need to adjust the password and group databases before entering the **protectvob** command.

If one or both of the replicas should not be ownership-preserving, follow these steps:

1. Use the **multitool chreplica** command to change the receiving replica to non-ownership-preserving.

   multitool chreplica –npreserve boston_hub@/vobs/dev
   ```
   Updated replica information for "boston_hub".
   ```

2. Import the packet.

   multitool syncreplica –import –receive
   ```
   Applied sync. packet
   /usr/atria/shipping/ms_ship/incoming/sync_sanfran_hub_18-Jan-00.16.54.14_3
   86_1 to VOB /net/minuteman/vobstg/dev.vbs
   ```

3. Change the status of the replicas.

   - If the sending replica should be non-ownership-preserving, change it to non-ownership-preserving.
   - If you want to retain ownership preservation in the receiving replica, change it back to ownership-preserving.

4. Export update packets from the sending and receiving replicas to all other replicas in the VOB family.

To avoid this problem in the future, use the procedure described in the section *Replica Permission Strategy* on page 40.

Object Mastership

An object mastered by one replica can depend on an object mastered by another replica. For example, an element and one of its subbranches are dependent objects, but these objects can be mastered by different replicas. As a result, certain kinds of inconsistent changes can be made at different replicas. The inconsistency is detected by **syncreplica –import**, causing it to fail with a recoverable error.

For example, if a type object is deleted in another replica, the replica at your site may refuse to import this change because a trigger type in the replica at your site depends on the deleted type object. During import, the following error message is displayed:

```
Can't delete attribute type type-name because of references to it in trigger
type restriction lists
```

1. If the trigger at your site is useful only with the deleted type object, use **cleartool rmtype trtype:***type-name* to delete the trigger type. Otherwise, replace the trigger type (**cleartool mktrtype –replace**) with a revised definition that does not depend on the deleted type object.

2. Reenter the **syncreplica –import** command.

Automatic Renaming of Type Objects and Replica Objects

The **syncreplica –import** command resolves naming conflicts among type objects or replica objects created at two or more replicas. For example, a *branch type* object named **v1.0_bugfix** is created at two different replicas. At some point, an invocation of **syncreplica –import** detects the conflict. (This may occur at one of the replicas that created the branch types, or at some other replica.)

syncreplica –import resolves the conflict by renaming the incoming object. In this example, branch type **v1.0_bugfix** is renamed to **boston_hub:v1.0_bugfix**, indicating that replica **boston_hub** is the master of the incoming type. **syncreplica –import** displays the following message:

```
multitool: Warning: To avoid name conflict,
generated name "boston_hub:v1.0_bugfix" ...
```

Intervention is not required at this point unless branch types or replicas are renamed. (Renaming of branch types affects the validity of config specs, and renaming of replicas may affect synchronization scripts.) However, if you do not rename the objects, different replicas have different names for the same object. In this example, the **boston_hub** replica calls a branch type **v1.0_bugfix**, but at least one other replica calls the same type object **boston_hub:v1.0_bugfix**.

The various sites involved in such a conflict must coordinate the renaming of all the objects involved, to guarantee that all objects have the same name in all replicas. Here is a general procedure:

1. The administrators at the sites decide how to rename the objects.

2. At the master replica of each type object or replica object, the administrator renames the type object or replica object.

   a. The Boston administrator renames the branch type that was created at the **boston_hub** replica:

      cleartool rename brtype:v1.0_bugfix v1.0_bugfix-boston_hub

   b. The San Francisco administrator renames the branch type that was created at the **sanfran_hub** replica:

      cleartool rename brtype:v1.0_bugfix v1.0_bugfix-sanfran_hub

   c. The Bangalore administrator renames the branch type that was created at the **bangalore** replica:

      cleartool rename brtype:v1.0_bugfix v1.0_bugfix-bangalore

3. All sites exchange update packets to propagate the name changes.

   NOTE: The name that caused the original conflict can be reused. One replica (and only one) can change the name to its original value:

   cleartool rename brtype:boston_hub:v1.0_bugfix v1.0_bugfix

   When this change is propagated to other replicas, it undoes any previous conflict-avoidance name changes, for example, by renaming **boston_hub:v1.0_bugfix** to **v1.0_bugfix**. (The propagation of this change must wait until after the other **rename** commands have been run in the other replicas and propagated throughout the VOB family, to make the name **v1.0_bugfix** available again.)

10.6 Running epoch_watchdog

If a VOB replica is restored improperly from backup, divergence can occur in the VOB family. When you restore a replica from backup, its epoch row is rolled back. If you do not run the **restorereplica** command on the replica before resuming development in the replica, divergence can occur.

For example, oplogs 1-700 are created in a replica and exported to sibling replicas. The replica is then restored from backup and its epoch number becomes 600 (operations 601-700 occurred after the backup copy was created). If the administrator does not run the **restorereplica** command, development resumes and new oplogs are created starting with ID 601. These oplogs have the same ID as the oplogs that were exported to other replicas before the restoration, but the operations themselves are different. The restored replica has diverged from the other replicas.

The **epoch_watchdog** script checks whether a VOB replica's epoch numbers have rolled back without a **restorereplica** command being run. We recommend that you run this script regularly as a scheduled job on all replica server hosts. For example, the following job runs **epoch_watchdog** every three hours for all VOBs on the host:

```
Job.Begin
  Job.Id: 20
  Job.Name: "epoch_watchdog"
  Job.Description.Begin:
  Run epoch_watchdog for each replicated VOB on this host.
  Job.Description.End:
  Job.Schedule.Daily.Frequency: 1
  Job.Schedule.StartDate: 3-Sep-2001
  Job.Schedule.FirstStartTime: 20:00:00
  Job.Schedule.StartTimeRestartFrequency: 03:00:00
  Job.DeleteWhenCompleted: FALSE
  Job.Task: 105
  Job.Args: -all
  Job.NotifyInfo.OnEvents: JobEndOKWithMsgs,JobEndFail
  Job.NotifyInfo.Using: email
  Job.NotifyInfo.Recipients: ms_admin
Job.End
```

This job uses the **MultiSite Epoch Watchdog** task, which is defined as follows:

UNIX task:

```
Task.Begin
    Task.Id:       105
    Task.Name:     "MultiSite Epoch Watchdog"
    Task.Pathname: epoch_watchdog
Task.End
```

Windows task:

```
Task.Begin
    Task.Id:       105
    Task.Name:     "MultiSite Epoch Watchdog"
    Task.Pathname: epoch_watchdog.bat
Task.End
```

For more information about creating tasks and scheduling jobs, see the **schedule** reference page in *Command Reference* and the *Administrator's Guide* for Rational ClearCase.

10.7 Restoring and Replacing Replicas

Occasionally, a VOB storage directory is lost. This can occur because of a hardware failure (for example, disk crash), a software failure (for example, OS-level file-system corruption), or a human error (for example, an **rm –fr** or **del** command). If an unreplicated VOB storage directory is lost, you can restore a recent copy from backup and resume development work. The changes made between the time of the backup and the time of the failure are not recoverable.

Similarly, if you lose the storage directory of a replicated VOB (that is, the storage for the replica used by developers at your site), you can restore a recent copy from backup. But matters are more complicated:

- Some of the work done between the time of the backup and the time of the failure may be recoverable. If some of the operations were sent to other replicas in update packets, these operations must be retrieved and imported.
- The restored copy of the replica is out of date. You must make this replica consistent with the other replicas in the VOB family before development can proceed at your site. Failure to reestablish consistency can lead to irreparable damage.

Because this procedure involves substantial effort, it is intended for situations where serious damage has occurred. (For example, the disk containing a replica is unusable.)

The method you use to restore the replica depends on how you back it up:

- If you lock your primary replica to back it up, you must restore it from the backup medium and perform the **restorereplica** procedure. See *Restoring a Replica from Backup*.
- If you never lock your primary replica and rely solely on a replica at your site as backup, you must replace the replica completely. See *Replacing an Existing Replica* on page 192.

Restoring a Replica from Backup

To restore a replica from backup:

1. Follow the procedure in the *Administrator's Guide* for Rational ClearCase to load the backup copy of the VOB storage directory.

2. As the VOB owner, **root** user (on UNIX) or a member of the ClearCase administrators group (on Windows), run the special MultiSite command to restore the replica:

   multitool restorereplica –invob *vob-selector*

   This places a special lock on the VOB object, which is in addition to the ClearCase lock created during the backup process. Between this point and the completion of Step #7, the **syncreplica –import** command adjusts the ClearCase lock temporarily to permit application of the update, then restores the full lock. During this time, only **syncreplica –import** can modify the replica.

3. Verify that all update packets have been processed at their destination replicas.

4. (Applicable only if the replica you're restoring was used to create one or more new replicas between the time of the backup and the time of the failure, and the other replicas in the family do not have information about the new replicas) The new replicas are unknown to your restored replica and all other replicas in the family, and **lsreplica** does not list them. If this is the case:

   a. At each new replica, set the estimated states of the siblings to their actual states. (Use **chepoch –actual** or **lsepoch**/**chepoch**. See *Recovering from Lost Packets* on page 182.)

   b. At each new replica, export update packets to all other replicas in the family except the restored replica.

   c. Import the packets exported in Step #b.

5. At the restored replica, generate update packets for all other replicas, and send the packets to the sibling replicas.

   You can send the packets using your standard synchronization method. To recover the replica more quickly, create the packets with **syncreplica –export –fship**.

   Because your replica is in the special restoration state, each outgoing update packet includes a special request for a return acknowledgment. It also includes your replica's old epoch

numbers, which are now its current epoch numbers, by virtue of the restoration backup in Step #1. Each destination replica uses these numbers to roll back its row for your replica.

6. Wait for each other replica in the VOB family to send an update packet to the restored replica. As in Step #5, you can accelerate the creation and delivery of the update packets.

   Collectively, these update packets include all the operations that occurred between the time of the backup and the last update that your replica sent out before its storage was lost—even operations that originated at your replica. (The packets also include more recent operations that originated at other replicas.) In addition, each incoming packet includes the requested return acknowledgment from the sending host.

7. Process the incoming update packets with **syncreplica –import**. When your replica has received return acknowledgments from all other replicas in the VOB family, **syncreplica –import** reports that restoration of the replica is complete:

   ```
   VOB has completed restoration: ...
   ```

8. (Applicable only if you had to perform Step #4) At one of the replicas that did not have information about the new replicas before the restoration procedure, export update packets to all of the new replicas and import the packets at the new replicas. (Do not perform this export from the restored replica.)

9. Unlock the VOB object in the restored replica.

   cleartool unlock vob:*pname-in-vob*
   ```
   Unlocked versioned object base "VOB-tag".
   ```

Development work in the replica can now resume.

Replacing an Existing Replica

If you must replace an existing replica, you can re-create it from one of the other replicas in the VOB family. For example, if you use Rational ClearCase MultiSite as your only backup mechanism and you must restore from a backup replica, you have to replace the working replica.

In this procedure, "backup replica" refers to the replica from which you restore the lost or deleted replica. If you have multiple replicas in the VOB family and you use more than one as a backup, use the replica that has most recently imported an update packet from the lost replica.

CAUTION: Do not use this procedure to fix import failures unless you have tried all other solutions, and Rational Technical Support advises you to follow these steps.

To replace a replica, use the following procedure (assume **boston_hub** on host **minuteman** is to be replaced, and **sanfran_hub** and **bangalore** are the other replicas in the VOB family):

1. For all views that use **boston_hub**, use the **lsprivate** command to list view-private and checked-out files. (To list views for which the VOB holds objects, use the **cleartool describe vob:** command.)

2. Check in all files (if possible) and save copies of view-private files out of the view. If you plan to save the views, use the procedure in *Saving Views from the Replaced Replica* on page 195 at this point.

3. If boston_hub can export update packets:

   a. On host **minuteman**, send update packets to **sanfran_hub** and **bangalore** from **boston_hub**:

      multitool syncreplica –export –fship sanfran_hub bangalore

   b. On the hosts where **sanfran_hub** and **bangalore** physically reside, import the packet from **boston_hub**:

      multitool syncreplica –import –receive

4. Back up **boston_hub**'s VOB storage to a storage medium.

5. At **sanfran_hub**, create a new replica, **boston_hub2**.

   multitool mkreplica –export –workdir /tmp/create –nc –fship minuteman:boston_hub2

6. If you did not use the **–fship** option in Step #5, transport the replica-creation packet to the host **minuteman**.

7. Create the new replica. On host **minuteman**:

   a. Unregister and remove the VOB-tag for **boston_hub**:

      cleartool umount /vobs/dev
      cleartool unregister –vob /net/minuteman/vobstg/dev.vbs
      cleartool rmtag –vob /vobs/dev

   b. Import the packet you created in Step #5 (include any special options you need):

   **multitool mkreplica –import –workdir /tmp/ms_wkdir –tag /vobs/dev2 **
   **–vob /net/minuteman/vobstg/dev2.vbs –nc –preserve –vrep boston_hub2 **
   /var/adm/atria/shipping/ms_ship/incoming/sh_o_repl_sanfran_hub_18-May-99.15:50:00_1

c. Mount **dev2**:

   cleartool mount /vobs/dev2

8. Make sure that **boston_hub2** can synchronize successfully:

   a. Set a view, change to a directory in **/vobs/dev2**, and generate a new label or attribute type. (Use a new view, not an old one that may have been used in **boston_hub**.)

   b. Create and send update packets to **sanfran_hub** and **bangalore**:

      multitool syncreplica –export –fship sanfran_hub bangalore

   c. At **sanfran_hub** and **bangalore**, import the update packet:

      multitool syncreplica –import -receive

   d. At **sanfran_hub** and **bangalore**, list the new type created in Step #a:

      cleartool lstype *type-selector*

9. Transfer mastership of all objects in **boston_hub** to **boston_hub2**.

   a. Determine which replica masters **boston_hub**.

   b. If **boston_hub** masters itself, run the following command at **boston_hub2**; if another replica masters **boston_hub**, run the following command at that replica:

      multitool chmaster –all –obsolete_replica boston_hub boston_hub2

   c. If **boston_hub** did not master itself, send an update packet from the master replica to **boston_hub2** and import it.

10. Make sure that **sanfran_hub**, **bangalore**, and **boston_hub2** can export and import update packets successfully.

11. At the site that masters **boston_hub**, remove the replica object for **boston_hub**:

    multitool rmreplica boston_hub

12. Synchronize all replicas in the family.

13. Remove the physical storage for **boston_hub** with standard operating system commands.

14. Remove the views that were used in **boston_hub**. (If you want to keep these views, use the procedure in *Saving Views from the Replaced Replica*.)

Saving Views from the Replaced Replica

To save the views used in the replaced replica:

1. Move all view-private files into the view's **lost+found** directory (*replica-uuid* is **boston_hub**'s UUID):

   cleartool recoverview –vob *replica-uuid* **–tag** *view-tag*

2. List view-private files in each of the views:

   cleartool lsprivate –tag *view-tag* **–invob** *vob-selector*

3. Use the **uncheckout** command to cancel all checkouts in the replica to be replaced; use the **–keep** option to save copies of the files.

4. Copy the **.keep** files to temporary directories outside the view. You can refer to these files when the new replica is available and you've checked out the elements again.

5. Use the **rmdo** command to remove all derived objects associated with the VOB to be replaced.

6. Remove all **.cmake.state** files.

7. Decide whether any valuable information is in any of the other view-private files associated with the VOB to be replaced.

After the replacement replica is back online, complete these additional steps:

1. Rebuild all derived objects.

2. Reconcile view-private files.

   Because view-private files are associated with a particular replica, restoration from backup makes them inaccessible. To continue work on checkouts, you must determine all checkouts, capture the related files, and place them in the correct location.

   You can do this by implementing a view backup procedure for files that cannot be re-created easily. For example, write a script that uses the **lsprivate** command to find all view-private objects (except for derived objects) and back them up to a backup tree. If the structure of this tree mirrors the VOB structure, it is easier to put the files back in their correct locations.

3. Run the **recoverview** command to free space associated with view-private files for the replica you removed.

An alternative method is based on **recoverview**. After letting **recoverview** move private files to the view's **lost+found** directory, the moved files are captured and placed into a location appropriate for the new replica. The main problem with this method is that the file names **recoverview** generates are leaf names; any directory structure is lost.

4. Redo changes to pool assignments.

   Pool assignments are local to a replica, so re-creating the original replica may undo changes made to them. Major changes to pool structure must be duplicated manually at the backup replica.

10.8 Cleaning Up from Accidental Deletion of a Replica

This situation is a more catastrophic variation of the problem described in *Restoring a Replica from Backup* on page 191: a replica's storage directory is lost, and there is no backup to be restored. The procedure for handling this situation is similar to that in *Deleting a Replica* on page 119.

Perform this procedure in the replica that is the master of the deleted replica. (If the replica was its own master, perform this procedure in the replica that will assume mastership of the deleted replica's objects.) It is also important that the replica know about all the objects that were mastered by the deleted replica.

1. Transfer mastership of all the objects that were mastered by the deleted replica. For example, if replica **tokyo** is deleted, enter this command at replica **sanfran_hub**:

   multitool chmaster –all –obsolete_replica tokyo@/vobs/dev –long sanfran_hub

   CAUTION: Incorrect use of **–all –obsolete_replica** can lead to irreparable inconsistencies among the replicas in a VOB family.

2. Remove the VOB-replica object for the deleted replica.

   multitool rmreplica tokyo@/vobs/dev

3. Send an update packet to all other replicas in the VOB family, to inform them of the mastership changes and the replica deletion.

   multitool syncreplica –export ...

Using MultiSite for Backup and Interoperability

11 Backing Up VOBs with MultiSite

This chapter describes two ways to use Rational ClearCase MultiSite as a VOB backup strategy:

- Using a replica as a backup VOB to avoid locking a VOB
- Using multiple replicas to provide incremental backups

Using multiple replicas in a local area network may help with reliability, availability, performance, and backup strategy. However, recovery issues limit how easily and rapidly clients may be switched from one replica to another. The details of the recovery process are described in *Restoring and Replacing Replicas* on page 190.

Using MultiSite for backups means that the backup replica needs to remain online so that it can be updated frequently from the original. Almost twice as much disk space is required (you do not need exactly twice as much space, because derived objects are not replicated and the cleartext pool for the backup replica is smaller or nonexistent). Also, you need a MultiSite license as well as a ClearCase license for each developer who accesses the replicated VOB.

11.1 Using a Backup Replica

To back up a VOB consistently, the ClearCase administrator must lock the VOB. However, many sites cannot find convenient times to lock the VOB so that the lock does not interfere with development work. One solution is to use MultiSite to create a replica of a VOB in the same local area network as the original. Updates from the original VOB to the backup replica are scheduled to match the recovery characteristics desired, that is, how much development work your site can afford to lose. At backup time, the backup replica is locked and backed up, thereby not interfering with development work at the original VOB.

Handling Objects That Are Not Replicated

The most important thing to note is that a MultiSite replica is not a complete copy of a VOB; the following objects are not replicated, and therefore are not restored from backup:

- Derived objects

  After a recovery from backup, developers must rebuild derived objects associated with the VOB. Checked-in derived objects are replicated, so they are backed up.

- Triggers

  To make sure you can re-create triggers after a restoration from backup, you must record information about all triggers in a VOB replica. For example, use the command **lstype –kind trtype** to list all triggers in a VOB.

- Nonobsolete locks

  As with triggers, you must record information about nonobsolete locks. You can write shell scripts that capture and re-create the trigger and lock information.

Also, pool assignments are specific to a replica, so re-creating the replica from a backup replica can undo changes made to them. If you make major changes to a VOB's pool structure, use the **chpool** command to duplicate these changes at the backup replica. (At replica creation, you can also use the **–pooltalk** option with **mkreplica –import** to make pool assignments.)

Designing Synchronization Strategy

You must determine the frequency and direction of synchronization. Typically, synchronization occurs in one direction only; that is, the backup replica never sends packets to the development replica, except during restoration.

Frequency of synchronization depends on your development environment. Some sites synchronize every 24 hours, but sites with rapid development may synchronize every 15 minutes.

11.2 Using Replicas with Incremental Backup

When you use a replica as an incremental backup of a VOB, you still back up the original VOB. You set up a replica of the original VOB in the same local area network, and schedule frequent unidirectional synchronizations. If you restore the original VOB from backup, the replica serves as an incremental backup by supplying changes made since the last backup.

This strategy reduces the frequency of backups at the original replica. It avoids some of the procedures described in *Restoring a Replica from Backup* on page 191, but still requires saving information about triggers, locks, and major pool changes. It also has the same limitations as unreplicated recovery from backup: a view and a VOB may not be consistent with each other after ClearCase recovery. It can, however, reduce the frequency of backups enough to fit into normal maintenance schedules.

The backup replica must be registered in its own region.

11.3 Restoring a Replica from Backup

Use the procedure described in *Restoring a Replica from Backup* on page 191.

12 Using MultiSite for Interoperability

You can use multiple replicas in local area networks to provide native access to VOBs in a heterogeneous network. This chapter describes ClearCase and MultiSite support for multiple replicas in a LAN and gives setup instructions.

12.1 Advantages and Disadvantages

Advantages of using Rational ClearCase MultiSite for interoperability:

- No purchase or maintenance of NFS or SMB software.
- Replicas can be used in backup strategies.
- User and group IDs do not have to match across platforms.

Disadvantages of using MultiSite for interoperability:

- You must configure and maintain MultiSite synchronization.
- VOB servers are needed on both UNIX and Windows systems.
- Each platform must master its own branch; alternatively, mastership can be transferred.
- Changes made on each platform must be imported and merged on the other.
- Replicas cannot preserve ownership.

12.2 Restrictions on Multiple Replicas in a LAN

You must observe these restrictions when using multiple replicas in a local area network:

- Do not register multiple replicas of a VOB family in a single region.

  This restriction prevents multiple replicas from being mounted on a host and prevents developers from accessing multiple replicas of a VOB family with a single view.

- Locate cross-VOB symbolic links in branched directories.

  NOTE: If the leaf name of the UNIX VOB-tag is the same as the Windows VOB-tag (for example, **/vobs/dev** and **\dev**), this restriction does not apply.

  Cross-VOB symbolic links point to particular replicas. To make it possible for clients to use a different replica, you can branch the directory that contains the symbolic link. Branching the directory may lead to partitioning replica use based on projects.

  For example, assume a project uses the branch **v2.0_integration** as the integration branch and the directory **vob_links** contains all the symbolic links that cross VOBs. The project manager creates a **v2.0_integration** branch of the directory **vob_links**, and then adjusts any symbolic links to point to the VOB-tag of the replica in use for that project. For example, on UNIX:

  ls –l
  ```
  tests -> ../../tests
  gui_src -> ../../gui_src
  design -> ../../design
  ```

  On Windows:

  cleartool ls
  ```
  tests -> ../../tests
  gui_src -> ../../gui_src
  design -> ../../design
  ```

  The leaf name of the VOB-tag of the local replica is **gui_src_replica2**, so the project manager adjusts the symbolic links as follows:

  cleartool checkout –nc .
  cleartool rmname gui_src
  cleartool ln –s ../../gui_src_replica2 gui_src
  cleartool checkin .

This ensures that the correct replica is referenced during a build of this project.

You can also use one symbolic link that refers to another VOB and have other symbolic links refer to it. For example:

```
rational_install -> ../../vobs/rational/install
release_list -> rational_install/release_list
```

This limits the number of duplicate links that must be maintained. We also recommend that you avoid cross-VOB symbolic links as much as possible.

➤ Make sure case-sensitivity and text mode settings are correct.

You must make sure that case-sensitivity and the text mode are handled properly. If there are case conflicts among files at different replicas, errors occur during synchronization. The text mode controls the use of line terminators in files; differences in use of line terminators between UNIX and Windows editors cause unexpected behavior during file comparisons and merges.

The *Administrator's Guide* for Rational ClearCase describes how to handle case-sensitivity and text mode setup. Be sure to read it carefully before creating UNIX and Windows replicas.

CAUTION: Do not use MultiSite to create multiple copies of a VOB in a single ClearCase region. Because the VOB UUID is identical for all replicas in a VOB family and is stored in many structures within a VOB, there is no way to make the copy of the VOB unique. Creating and using multiple copies of a VOB in a single region causes **clearmake** and views to exhibit unpredictable behavior, may cause data loss, and is not supported by Rational Software.

12.3 Setting Up Multiple Replicas at One Site

This section describes the process of creating replicas at one site.

Creating a replica of an existing VOB doesn't split the storage. On the contrary, the new replica requires additional disk space to accommodate another complete copy of the VOB's database and storage pools. For information about splitting a VOB, see the *Administrator's Guide* for Rational ClearCase.

If both replicas are on UNIX hosts or in the same Windows domain, they can be *ownership-preserving*. Any change in the owner, group, or access mode of an element at one of the replicas is propagated to the other replica.

The following procedure creates a Windows replica from a UNIX replica:

1. On the UNIX host:

   **multitool mkreplica –export –work /tmp/ms_wkdir –fship **
   –c "create replica for Windows use" aquarium:boston_windows

   ```
   Generating replica creation packet <outgoing-packet-pathname>
    - shipping order file is <shipping-order-pathname>
   Dumping database...
     . . .
   Dumper done.
   Attempting to forward/deliver generated packets...
     -- Forwarded/delivered packet <outgoing-packet-pathname>
   ```

2. On the Windows host:

   multitool lspacket –short

   ```
   c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\incoming\repl_bos
   ton_hub_21-Jul-00.18.42.01_1
   ```

   multitool mkreplica –import –npreserve –work c:\tmp\msite ^
   -tag \dev –public –vob \\aquarium\vobs\dev.vbs ^
   c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\incoming\repl_boston_
   hub_21-Jul-00.18.42.01_1

   ```
   The packet can only be used to create replica "boston_windows"
     - VOB family is ecf68c58.90fe11cd.a393.08:00:09:49:29:cd
     - replica OID is 9947c591.912d11cd.a4b1.08:00:09:49:29:cd
   Should I create this replica? [no] yes
   Comments for "boston_windows":
   provide native Windows access to VOB
   .
   Processing packet
   c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\incoming\repl_bos
   ton_hub_21-Jul-99.18.42.01_1
   Loading database...
     . . .
   Loader done.
   Vob tag registry password:   <enter password> (password required to create public VOB)
   Registering VOB mount tag "\dev"...
   VOB replica successfully created.
   Host-local path: aquarium:C:\vobs\dev.vbs
   Global path:     \\aquarium\vobs\dev.vbs
   VOB ownership:
     owner susan
     group user
   ```

MultiSite Reference Pages

13 MultiSite Reference Pages

This section of the *Administrator's Guide* for Rational ClearCase MultiSite contains MultiSite reference pages.

apropos

Displays MultiSite command information or glossary terms

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX

SYNOPSIS

- Display command information:

  apr·opos *topic* ...

- Display glossary terms:

  apr·opos –glo·ssary [*topic-args*]

DESCRIPTION

This command displays information about MultiSite commands or ClearCase and MultiSite glossary definitions. Use **apropos** as you use the standard UNIX **whatis(1)** or **apropos(1)** command.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

Default: **apropos** uses the standard MultiSite **whatis** file, which contains information about MultiSite commands.

topic ...
: **apropos** searches for each *topic* character string in the standard MultiSite **whatis** file. The string can occur anywhere within the line.

–glo·ssary [*topic-args*]
: Combines all arguments into a single character string, and displays all definitions in the ClearCase and MultiSite glossary whose glossary terms include this character string. To display the entire glossary, omit the *topic-args* argument.

EXAMPLES

- Search for lines with the word "epoch" in the standard MultiSite **whatis** file.

  multitool apropos epoch

  ```
  chepoch                  Changes epoch numbers
  epoch_watchdog           Checks whether a replica's epoch numbers have
  rolled back when the replica is not in restoration mode
  lsepoch                  Displays epoch information
  recoverpacket            Resets epoch row table so changes in lost packets
  are resent
  ```

- Search for glossary terms that include the string "epoch".

  multitool apropos –glossary epoch

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

  +++ epoch number matrix
  (MultiSite) A complete set of epoch numbers, indicating the 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.
  ```

- Search for glossary terms that include the phrase "update packet".

  multitool apropos –glossary update packet

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

FILES

ccase-home-dir/**doc/man/ms_whatis**
ccase-home-dir/**doc/man/ms_whatis.aux**

SEE ALSO

In the *Command Reference*: **help**, **man**

chepoch

Changes epoch numbers

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

chepoch [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
{ [**–f·orce**] *replica-selector* [*replica-selector=value* ...] [*oid*=*value* ...]
| **–actual** [**–raise_only**] *sibling-replica-selector* ... }

DESCRIPTION

This command changes, in one replica, one or more of the epoch numbers that represent the replica's record of the states of other replicas. You cannot change a replica's own row because these epoch numbers record the actual state of the replica.

With **–actual**, **chepoch** contacts sibling replicas, retrieves their own epoch rows, and changes their rows in the current replica's epoch number matrix. This brings the current replica's epoch number matrix up to date with changes made at the sibling replicas. **chepoch –actual** works only between sites that have IP connections. If **chepoch** cannot contact a sibling replica, it prints an error and tries to contact the next replica you specified.

chepoch –actual detects whether the sibling replica or the current replica is missing oplog entries. If oplog entries are missing, the command prints one of the following messages:

```
Your replica ("replica-name") has fewer oplog entries for itself than
"replica-selector" has for your replica.
To avoid permanent data loss, your VOB administrator must initiate the
documented replica restoration procedure.
```

```
The replica "replica-name" has more oplog entries for "replica-selector" than
"replica-selector" has for itself.
To avoid permanent data loss, its administrator must initiate the documented
replica restoration procedure.
```

For more information about epoch numbers, see *VOB Operations and the Oplog* on page 24. For descriptions of scenarios using **chepoch**, see *Cannot Find Oplog* on page 168 and *Lost Update Packet* on page 182.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB.

Mastership: No mastership restrictions.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SUPPRESSING INTERACTIVE PROMPTS. *Default:* Unless you specify **–actual**, you must confirm each epoch number change.

–f·orce
Suppresses confirmation steps.

SPECIFYING THE ROW TO BE CHANGED. *Default:* None. You must specify a replica.

replica-selector
Specifies the replica whose epoch row is to be changed; that is, changes the current replica's estimate of the state of *replica-selector*. Specify *replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*]

replica-name	Name of the replica (displayed with **lsreplica**)
vob-selector	VOB family of the replica; can be omitted if the current working directory is within the VOB. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

SPECIFYING THE CHANGES. *Default:* **chepoch** reads a set of *replica-selector=value* or *oid=value* pairs, one per line, from standard input. You can copy and paste **lsepoch** output, or type the data in the format described below. Extra white space is allowed. To terminate input, type a period character (**.**) and a carriage return (**<CR>**) at the beginning of a line.

replica-selector=value
oid=value

One or more arguments, where

replica-selector Column of the epoch number matrix. This argument, along with the preceding *replica-selector* argument, specifies a particular location in the matrix.

oid Object identifier for the replica. **lsepoch** prints OIDs in its output.

value New epoch number to be entered at the specified matrix location.

SETTING A ROW USING THE REPLICA'S ACTUAL STATE. *Default*: None. You must specify a replica.

–actual [**–raise_only**] *sibling-replica-selector* ...

Contacts *sibling-replica-selector*, retrieves its actual state, and changes its row in the epoch number matrix of the current replica. Specify *sibling-replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*] (see the description of *replica-selector*).

With **–raise_only**, **chepoch** raises epoch numbers for the sibling replica but does not lower any of them. This option optimizes synchronization when packets have been sent from the current replica to the sibling replica but have not yet been imported.

For example, replica **sanfran_hub** has received but not imported a packet from replica **boston_hub**. At replica **boston_hub**, the administrator uses **chepoch –actual** to reset the epoch row for **sanfran_hub** and then sends another update packet to **sanfran_hub**. This packet contains all the operations in the packet waiting to be imported at **sanfran_hub**, plus any new operations. If the administrator uses **chepoch –actual –raise_only** instead, the new packet includes only the new operations.

EXAMPLES

- Change two columns in the current replica's row for the **bangalore** replica.

  multitool chepoch bangalore boston_hub=950 sanfran_hub=2000

  ```
  Change oplog ID in row "bangalore", column "boston_hub" to 950 [no] yes
  Change oplog ID in row "bangalore", column "sanfran_hub" to 2000 [no] yes
  Epoch row successfully set.
  ```

- Make the same change as in the preceding example, but bypass the confirmation steps.

  multitool chepoch –force bangalore boston_hub=950 sanfran_hub=2000
  ```
  Epoch row successfully set.
  ```

- Make the same change as in the preceding examples, specifying the changes as terminal input instead of as command-line arguments.

  multitool chepoch bangalore
  ```
  Enter specifications for epochs to change in row "bangalore"
  (one per line)
  ```
  oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950
  oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=2000
  .
  ```
  Change oplog ID in row "bangalore", column "boston_hub" to 950 [no] yes
  Change oplog ID in row "bangalore", column "sanfran_hub" to 2000 [no] yes
  Epoch row successfully set.
  ```

- Change an item in a replica's estimate of the state of the **sydney** replica, specifying the VOB family of the replica whose matrix is to be changed.

  multitool chepoch –force sydney@\vob3 buenosaires=800
  ```
  Epoch row successfully set.
  ```

- Set the current replica's estimate of the state of the **tokyo** replica to its actual state.

  multitool chepoch –actual tokyo@/vobs/tromba
  ```
  Entry for          boston_hub changed from:        1400 to        1300
  Entry for         sanfran_hub changed from:         985 to         950
  Entry for               tokyo changed from:        2562 to        2000
  ```

- Update the current replica's epoch numbers for replicas **boston_hub** and **sanfran_hub**.

  multitool chepoch –actual boston_hub@/vobs/dev sanfran_hub@/vobs/dev
  ```
  Entry for           boston_hub changed from:        1400 to        1300
  Entry for          sanfran_hub changed from:         985 to        1000
  ```

- Make the same change as in the previous example, but do not lower any of the numbers.

  multitool chepoch –actual –raise_only boston_hub@/vobs/dev sanfran_hub@/vobs/dev
  ```
  Entry for           boston_hub unchanged from:        1400
  Entry for          sanfran_hub   changed from:         985 to     1000
  ```

SEE ALSO

lsepoch, **recoverpacket**, **restorereplica**

chmaster

Transfers mastership of VOB-database object

APPLICABILITY

Product	Command type
ClearCase	cleartool subcommand
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

chmaster [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery**
| **–cqe·ach** | **–nc·omment**]
{ *master-replica-selector object-selector* ...
| [**–pname**] *master-replica-selector branch-or-element-pname* ...
| **–str·eam** [**–ove·rride**] *master-replica-selector stream-selector* ...
| **–def·ault** [**–pname**] *branch-pname* ...
| **–def·ault** *brtype-selector* ...
| **–all** [**–obsolete_replica** *old-replica-selector*]
[**–l·ong**] [**–vie·w** *view-tag*] *master-replica-selector*
}

DESCRIPTION

This command transfers the mastership of one or more objects from one VOB replica to another. Only the current replica is affected immediately; other replicas are notified of the mastership transfers through the normal exchange of update packets.

To limit use of this command to a certain set of users, you can create triggers. For more information, see *Managing Software Projects*.

SPECIFYING A VIEW CONTEXT

The **chmaster** command requires a view context. If you are not in a set view or working directory view on UNIX or a view drive on Windows, you can specify a view on the command line, as shown in the following table. If you specify a dynamic view, it must be active on your host.

NOTE: A view you specify in the **chmaster** command takes precedence over your current set view, working directory view, or view drive.

Argument	How to specify a view
object-selector *brtype-selector*	Use a view-extended pathname as the *vob-selector* portion of the argument. For example: **lbtype:LABEL1@/view/jtg/vobs/dev** **brtype:v1.0_bugfix@/view/jtg/vobs/dev** **lbtype:LABEL1@s:\dev** **brtype:v1.0_bugfix@s:\dev**
branch-pname *element-pname*	Specify *branch-pname* or *element-pname* as a view-extended pathname. For example: **/view/jtg/vobs/dev/cmd.c@@** **/view/jtg/vobs/dev/cmd.c@@/main** **s:\dev\cmd.c@@** **s:\dev\cmd.c@@\main**
master-replica-selector (for the **chmaster –all** variant)	Use the **–view** option or use a view-extended pathname as the *vob-selector* portion of the argument. For example: **–view jtg replica:boston_hub@\dev** **replica:boston_hub@/view/jtg/vobs/dev** **replica:boston_hub@s:\dev**

RESTRICTIONS

Identities: For all UCM objects except baselines, no special identity is required. For baselines and all other non-UCM objects, you must have one of the following identities:

- Object creator (except for replicas)
- Object owner (except for replicas)
- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: Restrictions depend on the kind of object:

Object whose mastership is changing	Locks on these objects cause the chmaster command to fail
Element	Element, element type, VOB

Object whose mastership is changing	Locks on these objects cause the chmaster command to fail
Branch	Branch, branch type, VOB
Type object	Type object, VOB
Hyperlink	Hyperlink type, VOB
Baseline	Baseline, VOB, replica, components associated with the baseline
Stream	Stream, activity
Component	Component, VOB, replica

Mastership: Your current replica must master the object. Using both **–all** and **–obsolete_replica** overrides this restriction, but you must not use the **–obsolete_replica** option except in special circumstances. (See the description of the **–all** option.)

Other: You cannot transfer mastership of a branch if the branch is checked out reserved or if it is checked out unreserved without the **–nmaster** option.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SPECIFYING THE OBJECTS. *Default:* None.

master-replica-selector object-selector ...
Transfers mastership of objects specified with *object-selector* to the VOB replica specified with *master-replica-selector*. Specify *master-replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*]

replica-name Name of the replica (displayed with **lsreplica**)

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

Specify *object-selector* in one of the following forms:

vob-selector	**vob:***pname-in-vob*	
	where	
	pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)
attribute-type-selector	[**attype:**]*type-name*[@*vob-selector*]	
branch-type-selector	[**brtype:**]*type-name*[@*vob-selector*]	
element-type-selector	[**eltype:**]*type-name*[@*vob-selector*]	
hyperlink-type-selector	[**hltype:**]*type-name*[@*vob-selector*]	
label-type-selector	[**lbtype:**]*type-name*[@*vob-selector*]	
hlink-selector	[**hlink:**]*hlink-id*[@*vob-selector*]	
oid-obj-selector	**oid:***object-oid*[@*vob-selector*]	
replica-selector	[**replica:**]*replica-name*[@*vob-selector*]	
baseline-selector	[**baseline:**]*baseline-name*[@*vob-selector*]	
component-selector	[**component:**]*component-name*[@*vob-selector*]	

[**–pname**] *master-replica-selector branch-or-element-pname ...*

Transfers mastership of the specified branches or elements to the VOB replica specified with *master-replica-selector*. A branch pathname takes the form *element-name*@@/*branch*..., for example, **cmdsyn.c@@/main/bugfix**, and an element pathname takes the form *element-name*@@, for example, **cmdsyn.c@@**. If *branch-or-element-pname* has the form of an object selector, you must include the **–pname** option to indicate that *pname* is a pathname.

–str·eam [**–ove·rride**] *master-replica-selector stream-selector ...*

Transfers mastership of the specified streams and their associated objects to the VOB replica specified with *master-replica-selector*. Specify *stream-selector* in the following form:

stream-selector	[**stream:**]*stream-name*[@*vob-selector*]

Use the **–override** option only if the **chmaster –stream** command fails. With **–override**, **chmaster** attempts to transfer mastership of objects whose mastership was not transferred during the original invocation of the command. For more information, see *Managing Mastership* in *Administrator's Guide* for Rational ClearCase MultiSite.

–a·ll [**–obsolete_replica** *old-replica-selector*] [**–l·ong**] [**–vie·w** *view-tag*] *master-replica-selector*

CAUTION: Incorrect use of the **–obsolete_replica** form of the command can lead to divergence among the replicas in a VOB family.

Transfers to *master-replica-selector* mastership of all objects that are located in and mastered by the current replica. (The **chmaster** command determines the current replica by using the *vob-selector* you specify as part of *master-replica-selector*. If you do not include a *vob-selector*, **chmaster** uses the replica containing the current working directory.)

If errors occur, the command continues. After finishing, it reports that not all mastership changes succeeded.

With **–long**, **chmaster** lists the objects whose mastership is changing.

With **–view**, **chmaster** uses the specified view as the view context.

With **–obsolete_replica**, **chmaster** transfers mastership of all objects in the replica specified with *old-replica-selector*. Also, **chmaster** associates nonmastered checkouts with the new replica. Use this form of **chmaster** only when replica *old-replica-selector* is no longer available (for example, was deleted accidentally). Before entering this command, you must make sure that *old-replica-selector* masters itself or is mastered by the replica that it last updated. Then, enter the **chmaster** command at the last-updated replica. You must also send update packets from the last-updated replica to all other remaining replicas in the VOB family. For more information, see the **rmreplica** reference page.

RETURNING MASTERSHIP OF BRANCHES TO DEFAULT STATE. *Default:* None.

–def·ault [**–pname**] *branch-pname* ...

Transfers mastership of *branch-pname* to the replica that masters the branch type. If *branch-pname* has the form of an object selector, you must include the **–pname** option to indicate that *branch-pname* is a pathname.

–def·ault *brtype-selector* ...

Removes explicit mastership of branches that are mastered explicitly by the current replica and are instances of the type *brtype*.

NOTE: You can use this command only at the replica that masters the branch type.

EXAMPLES

- At replica **boston_hub**, transfer mastership of label type **V1.0_BUGFIX** to the **sanfran_hub** replica.

  multitool chmaster sanfran_hub lbtype:V1.0_BUGFIX

  ```
  Changed mastership of "V1.0_BUGFIX" to "sanfran_hub"
  ```

- At replica **sanfran_hub**, transfer mastership of element **list.c** to the **sydney** replica.

  multitool chmaster sydney list.c@@
  ```
  Changed mastership of "list.c" to "sydney"
  ```

- At replica **sanfran_hub**, transfer mastership of the stream **v2.1.bl5** and its associated objects to the **boston_hub** replica.

  multitool chmaster –stream boston_hub@/vobs/dev stream:v2.1.bl5@/vobs/dev

- At the replica that is the master of replica **sanfran_hub**, make **sanfran_hub** self-mastering.

  multitool chmaster sanfran_hub replica:sanfran_hub
  ```
  Changed mastership of "sanfran_hub" to "sanfran_hub"
  ```

- At replica **buenosaires**, transfer mastership of branch **cache.c@@/main/v3_dev** to **boston_hub**.

  multitool chmaster boston_hub cache.c@@/main/v3_dev
  ```
  Changed mastership of branch "/vobs/dev/cache.c@@/main/v3_dev" to
  "boston_hub"
  ```

- For all objects mastered by the current replica, transfer mastership to **sanfran_hub**.

  multitool chmaster –all sanfran_hub
  ```
  Changed mastership of all objects
  ```

- Same as the preceding example, but have **chmaster** list each object whose mastership is changing, and specify a view context.

  multitool chmaster –all –long sanfran_hub@/view/jtg/vobs/dev
  ```
  Changed mastership of branch type sydney_main
  Changed mastership of label type SYDNEY_V2.0
  Changed mastership of replica sydney
  Changed mastership of all objects
  ```

- Return mastership of a branch to the replica that masters the branch type, and then remove its explicit mastership.

  At the replica that masters the branch:

  multitool describe –fmt "%[master]p\n" brtype:v3_bugfix
  ```
  boston_hub@\dev
  ```

  multitool chmaster boston_hub@\dev \dev\acc.c@@\main\v3_bugfix
  ```
  Changed mastership of branch "\dev\acc.c@@\main\v3_bugfix" to
  "boston_hub@\dev"
  ```

  multitool syncreplica –export –fship boston_hub@\dev
  ```
  Generating synchronization packet c:\Program Files\Rational\ClearCase\var
  \shipping\ms_ship\outgoing\sync_bangalore_19-Aug-00.09.33.02_3447_1
  ...
  ```

At the replica that masters the branch type:

multitool syncreplica –import –receive

```
Applied sync. packet
/var/adm/atria/shipping/ms_ship/incoming/sync_bangalore_19-Aug-00.09.33.02
_3447_1
to VOB /net/minuteman/vobstg/dev.vbs
```

multitool chmaster –default brtype:v3_bugfix

```
Changed mastership of branch type "v3_bugfix" to "default"
```

SEE ALSO

reqmaster, syncreplica
Chapter 8, *Managing Mastership* in the *Administrator's Guide* for Rational ClearCase MultiSite.

chreplica

Changes the properties of a replica

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

chrep·lica [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**] [**–hos·t** *hostname*]
[**–pre·serve** | **–npr·eserve**]
[**–isconn·ected** | **–nconn·ected**] *replica-selector*

DESCRIPTION

This command changes the properties of a VOB replica. For more information, see *Changing the Host Name for a Replica* on page 113 and *Changing Ownership Preservation* on page 114.

RESTRICTIONS

Identities: You must have one of the following identities:

- Creator of the replica where you enter the command
- Owner of the replica where you enter the command
- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB object, replica object.

Mastership: With **–isconnected** and **–nconnected**, there are no mastership restrictions. With all other options, your current replica must master the replica being changed.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SPECIFYING THE CHANGE. *Default:* None. You must specify at least one of **–host**, **–preserve**, **–npreserve**, **–isconnected**, or **–nconnected**.

–hos·t *hostname*
Changes the host name associated with the specified replica. *hostname* must be usable by hosts in different domains.

hostname can be either the IP address of the host or the computer name, for example, **minuteman**. You may have to append an IP domain name, for example, **minuteman.purpledoc.com**.

On UNIX, use the **uname –n** command to display the computer name. On Windows NT, the computer name is displayed in the **Network Settings** dialog box, which is accessible from the **Network** icon in the Control Panel. On Windows 2000, the computer name is displayed on the **Network Identification** tab in the **System Properties** dialog box, which is accessible from the **System** icon in the Control Panel.

–pre·serve
Makes the specified replica ownership-preserving.

–npr·eserve
Removes the specified replica from the set of ownership-preserving replicas.

–isconn·ected | **–nconn·ected**
Indicates whether the replica has IP connectivity to the current replica. You must specify a sibling replica; you cannot set this property for your current replica.

SPECIFYING THE REPLICA. *Default:* None.

replica-selector
Specifies the replica to be changed. Specify *replica-selector* in the form [**replica:**]*replica-name*[*@vob-selector*]

replica-name	Name of the replica (displayed with **lsreplica**)
vob-selector	VOB family of the replica; can be omitted if the current working directory is within the VOB. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

- Associate replica **bangalore** with host **ramohalli** in the database of the current replica.

  multitool chreplica –host ramohalli bangalore
  ```
  Updated replica information for "bangalore".
  ```

- Make replica **tokyo** a non-ownership-preserving replica.

  multitool chreplica –npreserve tokyo
  ```
  Updated replica information for "tokyo".
  ```

- Mark replica **sydney** as not connected.

  multitool chreplica –nconnected sydney
  ```
  Updated replica information for "sydney".
  ```

SEE ALSO

chmaster, **syncreplica**

epoch_watchdog

Checks whether a replica's epoch numbers have rolled back when the replica is not in restoration mode

APPLICABILITY

Product	Command Type
MultiSite	MultiSite command

Platform
UNIX
Windows

SYNOPSIS

- Check for rollback of epoch numbers:

  epoch_watchdog { **–all** | **–vobs** *VOB-tag*,... | *list-file* }

- Print help on command options:

  epoch_watchdog –help

On UNIX, **epoch_watchdog** is located in *ccase-home-dir***/config/scheduler/tasks**. On Windows, **epoch_watchdog** is located in *ccase-home-dir***\config\scheduler\tasks**.

DESCRIPTION

epoch_watchdog checks whether a VOB replica's epoch numbers have rolled back without a **restorereplica** command being run. If the epoch numbers have rolled back and the replica is not in restoration mode, the VOB may have been improperly restored from backup. This script is intended to be run regularly by the ClearCase scheduler. For more information, see the **schedule** reference page in the *Command Reference*.

epoch_watchdog writes a replica's epoch number to a log file in **/var/adm/atria/log/epoch_logs** on UNIX or *ccase-home-dir***\var\log\epoch_logs** on Windows. The next time the script is run, it compares the current epoch number to the logged number. If **epoch_watchdog** finds that the current number is lower than the logged number, it checks to see if the replica is in restore mode. If the replica is not being restored, **epoch_watchdog** attempts to lock the affected VOB, and optionally sends email notification (you must specify email addresses in the scheduled job). In this situation, you must contact Rational Support before unlocking the VOB or attempting any repair procedures.

NOTE: If the time window between scheduled jobs of **epoch_watchdog** is large enough, the activity level in the replica can be high enough that a rollback can go undetected in the case where restorereplica is not performed.

If an error occurs, **epoch_watchdog** creates an entry in **/var/adm/atria/log/error_log** on UNIX or the Event Viewer on Windows. This entry references the **epoch_logs** file.

RESTRICTIONS

You must be root on UNIX or a member of the ClearCase administrators group on Windows.

OPTIONS AND ARGUMENTS

–h·elp
Prints help on command options.

–all
Checks all replicated VOBs on the current computer.

–vobs *VOB-tag*,...
VOB-tags of replicated VOBs to be checked. Specify multiple VOB-tags in a comma-separated list with no white space.

list-file
Path to file containing a list of VOBs to check. Specify one VOB on each line, with no white space, in the following form:

vob:*VOB-tag*

EXAMPLES

FILES

/var/adm/atria/log/epoch_logs (UNIX)
/var/adm/atria/log/error_log (UNIX)
*ccase-home-dir***\var\log\epoch_logs** (Windows)

SEE ALSO

schedule (in the *Command Reference*)

lsepoch

Displays epoch information

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsepoch [**–invob** *vob-selector* | [**–actual**] *replica-selector* ...]

DESCRIPTION

By default, **lsepoch** displays the epoch number matrix of the VOB replica containing the current working directory. The replica's own epoch row represents its actual state. The other rows represent the replica's best estimate of other replicas' states.

NOTE: **lsepoch** output includes rows for replicas that have been deleted, in addition to the rows for replicas still in use. Oplog records for deleted replicas are saved in case a replica undergoing restoration must receive oplogs from the deleted replica. (For example, a replica may be restored from a backup created before the deleted replica was removed.)

With **–actual**, **lsepoch** contacts sibling replicas and retrieves their epoch rows. These epoch rows reflect the replicas' actual states. **lsepoch –actual** works only between sites with IP connections. If **lsepoch** cannot contact a sibling replica, it prints an error and tries to contact the next replica you specified. **lsepoch –actual** detects whether the sibling replica or the current replica is missing oplog entries. If oplog entries are missing, the command prints one of the following messages:

```
Your replica ("replica-name") has fewer oplog entries for itself than
"replica-selector" has for your replica.
To avoid permanent data loss, your VOB administrator must initiate the
documented replica restoration procedure.
```

```
The replica "replica-name" has more oplog entries for "replica-selector" than
"replica-selector" has for itself.
To avoid permanent data loss, its administrator must initiate the documented
replica restoration procedure.
```

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

–invob *vob-selector*

Displays the epoch number matrix of the current replica in the VOB family specified by *vob-selector*. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–actual

Retrieves epoch rows from sibling replicas.

replica-selector ...

Without **–actual**, displays the current replica's row for each specified replica. With **–actual**, contacts each specified replica and displays the replica's own epoch row. Specify *replica-selector* in the form [**replica:**]*replica-name*[*@vob-selector*]

replica-name Name of the replica (displayed with **lsreplica**)

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

- Display the epoch number matrix for the current replica in the VOB family **/vobs/dev**.

  cd /vobs/dev
  multitool lsepoch

```
For VOB replica "/vobs/dev":
Oplog IDs for row "bangalore" (@ ramohalli):
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=0          (bangalore)
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=950        (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=10         (sanfran_hub)
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=0          (bangalore)
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=1          (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=10         (sanfran_hub)
```

```
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=0          (bangalore)
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=1          (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=16         (sanfran_hub)
```

- Display the epoch number matrix for the current replica in the VOB family **\doc**.

  multitool lsepoch –invob \doc

```
For VOB replica "\doc":
Oplog IDs for row "boston_hub" (@ minuteman):
 oid:fb4d4850.093022d1.b033.00:50:98:97:24:76=836        (boston_hub)
 oid:lw5b4639.039011d1.b083.00:60:97:98:42:69=580        (sanfran_hub)
Oplog IDs for row "sanfran_hub" (@ goldengate):
 oid:fb4d4850.093022d1.b033.00:50:98:97:24:76=600        (boston_hub)
 oid:lw5b4639.039011d1.b083.00:60:97:98:42:69=785        (sanfran_hub)
```

- List the current replica's estimate of the state of replica **sydney**.

  multitool lsepoch sydney@/vobs/dev

```
For VOB replica "/vobs/dev":
Oplog IDs for row "sydney" (@ sanfran_hub):
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=0          (boston_hub)
 oid:0eaa6fc3.737d11d4.adbe.00:01:80:c0:4b:e7=1          (sanfran_hub)
 oid:c6b8c9b0.038d11d1.b083.00:60:97:98:42:69=16         (sydney)
```

- List the actual state of the **bangalore** and **buenosaires** replicas.

  multitool lsepoch –actual bangalore@/vobs/dev buenosaires@/vobs/dev

```
Contacting remote replica...
bangalore:
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=0          (bangalore)
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=20         (boston_hub)
 oid:ac93e6cf.14a311d5.bbcc.00:01:80:c0:4b:e7=950        (buenosaires)
Contacting remote replica...
buenosaires:
 oid:7ag3b0bc.defa11d0.ba57.00:01:72:73:3c:94=0          (bangalore)
 oid:87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7=16         (boston_hub)
 oid:ac93e6cf.14a311d5.bbcc.00:01:80:c0:4b:e7=950        (buenosaires)
```

SEE ALSO

chepoch, **recoverpacket**, **restorereplica**

lsmaster

Lists objects mastered by a replica

APPLICABILITY

Product	Command type
ClearCase	cleartool subcommand
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsmaster [**–kind** *object-selector-kind*[**,**...]] [**–fmt** *format-string*] [**–view** *view-tag*]
[**–inr·eplicas** { **–all** | *replica-name*[**,**...] }] *master-replica-selector* ...

DESCRIPTION

This command lists objects mastered by a replica. By default, the command uses only the information known to your current replica. If you list objects mastered by a sibling replica, changes that have not been imported at your current replica are not included in the output. For example, a label type is added at replica **sanfran_hub**, but replica **boston_hub** has not imported the update packet containing the change. If you enter the command **multitool lsmaster sanfran_hub** at the **boston_hub** replica's site, the output does not include the new label type.

To retrieve information from a sibling replica, use **–inreplicas**. This form of the command contacts the sibling replicas and works only between sites that have IP connections. If **lsmaster** cannot contact a replica, it prints an error and tries to contact the next replica you specified.

Object Name Resolution

If you have a view context, **lsmaster** uses the view to resolve object identifiers (OIDs) of file system objects to the names of the objects. If you do not have a view context, **lsmaster** prints OIDs for file system objects. You can specify a view context with the **–view** option.

When you specify **–inreplicas**, **lsmaster** prints OIDs for objects whose creation operations have not yet been imported at your current replica.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SPECIFYING THE OBJECT KINDS. *Default:* **lsmaster** lists all objects mastered by the replica.

–kind *object-selector-kind*[,...]

Limits the listing to the specified object kinds. The list of object kinds must be comma-separated, with no spaces. *object-selector-kind* can be one of the following values:

Values for ClearCase:

attype
branch
brtype
delem (directory element)
eltype
felem (file element)
hlink
hltype
lbtype
slink
vob

Values for ClearCase UCM:

activity
baseline
component
folder
project
stream

Values for MultiSite:

replica

REPORT FORMAT. *Default:* For file-system objects, the master replica, object kind, and OID of each object are listed. For example:

```
master replica: boston_hub@/vobs/dev  file element:oid:40e022a3.241d11ca ...
```

For non-file-system objects, the master replica, object kind, and name of each object are listed. For example:

```
master replica: boston_hub@/vobs/dev  brtype:main
```

–fmt *format-string*
Lists information using the specified format string. See the **fmt_ccase** reference page for details on using this option.

SPECIFYING A VIEW CONTEXT. *Default:* The command uses your current view context.

–view *view-tag*
Specifies a view.

SPECIFYING THE REPLICA FROM WHICH TO RETRIEVE INFORMATION. *Default:* The command uses the information in your current replica.

–inr·eplicas { **–all** | *replica-name*[,...] }
With **–all**, retrieves information from all replicas in the VOB family (except deleted replicas). Otherwise, retrieves information from the sibling replicas you specify. The list of replicas must be comma-separated, with no spaces.

SPECIFYING THE REPLICA WHOSE MASTERED OBJECTS ARE DISPLAYED. *Default:* No default; you must specify a replica.

master-replica-selector ...
Lists objects mastered by the specified replica. Specify *master-replica-selector* in the form [**replica:**]*replica-name*[*@vob-selector*]

replica-name Name of the replica

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

- List all objects mastered by the replica **sanfran_hub**.

  multitool lsmaster –view v4.1 –fmt "%m:%n\n" sanfran_hub@/vobs/dev

```
directory element:/vobs/dev.@@
...
file element:/vobs/dev/lib/file.c@@
...
symbolic link:/vobs/dev/doc
...
hyperlink:Merge@2@/vobs/dev
...
```

- List all label types mastered by the replica **boston_hub**.

  cleartool lsmaster –fmt "%m:%n\n" –kind lbtype boston_hub@\doc
  ```
  label type:LATEST
  label type:CHECKEDOUT
  label type:BACKSTOP
  label type:REL1
  ...
  ```

- List all element types, label types, and branch types mastered by the replica **sanfran_hub**.

  cleartool lsmaster –kind eltype,lbtype,brtype sanfran_hub
  ```
  master replica: sanfran_hub@\dev "element type" file_system_object
  master replica: sanfran_hub@\dev "element type" file
  master replica: sanfran_hub@\dev "element type" directory
  ...
  master replica: sanfran_hub@\dev "branch type" sanfran_main
  master replica: sanfran_hub@\dev "branch type" v1.0_bugfix
  ...
  master replica: sanfran_hub@\dev "label type" LATEST
  master replica: sanfran_hub@\dev "label type" SANFRAN_V2.0
  master replica: sanfran_hub@\dev "label type" V1.0_BUGFIX
  master replica: sanfran_hub@\dev "label type" TOKYO_BASE
  master replica: sanfran_hub@\dev "label type" SYDNEY_BASE
  ...
  ```

- List the name and creation comment of each element type mastered by the replica **boston_hub**. Contact the **boston_hub** replica to retrieve the data.

  **multitool lsmaster –inreplicas boston_hub –fmt "%n\t%c\n" **
  –kind eltype boston_hub@/vobs/dev
  ```
  In replica "boston_hub"
  binary_delta_file       Predefined element type used to represent a file
  in binary delta format.
  ...
  ```

- List information from all replicas in the VOB family about the objects mastered by the replica **sanfran_hub**. Do not use a view context.

  multitool lsmaster –inreplicas –all sanfran_hub@/vobs/dev
  ```
  In replica "boston_hub"
  master replica: sanfran_hub@/vobs/dev "versioned object base" /vobs/dev
  master replica: sanfran_hub@/vobs/dev "directory element"
  (oid:40e0000b.241d23ca.b3df.08:00:69:02:05:33)
  master replica: sanfran_hub@/vobs/dev "directory element"
  (oid:40e0000b.241d23ca.b3df.08:00:69:02:05:33)
  ...
  ```

Use a view context:

multitool lsmaster –view v4.1 –inreplicas –all sanfran_hub@/vobs/dev

```
In replica "boston_hub"
master replica: sanfran_hub@/vobs/dev "versioned object base" /vobs/dev
master replica: sanfran_hub@/vobs/dev "directory element"
/view/v4.1/vobs/dev/.@@
master replica: sanfran_hub@/vobs/dev "directory element"
/view/v4.1/vobs/dev/lib@@
...
```

- List information from the **sanfran_hub** replica about the objects mastered by the replica **boston_hub**.

  multitool lsmaster –view v4.1 –inreplicas sanfran_hub boston_hub@\doc

- List all projects, baselines, and streams mastered by the replica **boston_hub**. Contact the **boston_hub** replica to retrieve the data.

multitool lsmaster –inreplicas boston_hub –kind project,baseline,stream \ boston_hub@/vobs/projects

```
In replica "boston_hub"
master replica: boston_hub@/vobs/projects  "project" V4.5.BL3
master replica: boston_hub@/vobs/projects  "project" doc_localize
master replica: boston_hub@/vobs/projects  "stream" 4.5.bl2_int
master replica: boston_hub@/vobs/projects  "project" V4.5.BL2
master replica: boston_hub@/vobs/projects  "stream" 4.5.bl2
master replica: boston_hub@/vobs/projects  "stream" stream000317.160434
master replica: boston_hub@/vobs/projects  "stream" stream000317.173156
master replica: boston_hub@/vobs/projects  "baseline" V4.5.BL2.011005.12820
master replica: boston_hub@/vobs/projects  "baseline" V4.5.BL2.011005.12890
master replica: boston_hub@/vobs/projects  "baseline" V4.5.BL2.011005.17408
master replica: boston_hub@/vobs/projects  "baseline" V4.5.BL2.011005.17695
master replica: boston_hub@/vobs/projects  "baseline" V4.5.BL2.011005.19614
...
```

SEE ALSO

chmaster, **describe**, **reqmaster**
Introduction to MultiSite and *Managing Mastership* in the *Administrator's Guide* for Rational ClearCase MultiSite.

lspacket

Describes contents of packet

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

lspacket [**–l·ong** | **–s·hort**] [*pname* ...]

DESCRIPTION

This command lists a summary of the contents of one or more disk files that contain replica-creation or update packets. By default, the **lspacket** output includes this information:

- Pathname of each packet
- Type of each packet (`Replica Creation` or `Update`)
- VOB family to which the packet applies
- Creation comment for the packet
- Replicas for which the packet is intended; if the VOB-tag is available, **lspacket** displays it.

  An asterisk after a replica name indicates that the packet can be imported immediately because it does not depend on any other packet. (This applies only for replicas listed in the host's ClearCase registry.)

  For example, if there are two packets waiting to be imported at a replica, the first packet has an asterisk and the second doesn't (because the second packet depends on the first).
- Packet sequence number (for a disk file storing one part of a logical packet that has been split into multiple physical packets)

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

LISTING FORMAT. *Default:* Includes the information listed in the *DESCRIPTION* section.

–l·ong

In addition to the default information, lists the following information:

- Name or OID of the replica where the packet was created
- Oplog IDs (epoch numbers) that indicate the contents of the packet
- Recovery incarnation of the sending replica (an internal value used by MultiSite)
- Major and minor packet versions, which are values for use by Rational Software

–s·hort

Lists only the pathname of a packet.

SPECIFYING THE PACKETS TO BE LISTED. *Default:* Lists all packets in all storage bays on the current host.

pname ...

One or more pathnames of files and/or directories.

Each file you specify is listed if it contains a physical packet. For each directory you specify, **lspacket** lists packets stored in that directory.

EXAMPLES

- List a single replica-creation packet.

  **multitool lspacket **
  /usr/atria/shipping/ms_ship/incoming/repl_boston_15-Aug-00.17.07.20_7865_1

  ```
  Packet is:
  /usr/atria/shipping/ms_ship/incoming/repl_boston_15-Aug-00.17.07.20_7865_1
  Packet type: Replica Creation
  VOB family identifier is: 94be56a1.0dd611d1.a0df.00:01:80:7b:09:69
  Comment supplied at packet creation is:
  Packet intended for the following targets:
    buenosaires
  The packet sequence number is 1
  ```

- List a single update packet.

  multitool lspacket /usr/tmp/packet1

  ```
  Packet is: /usr/tmp/packet1
  Packet type: Update
  VOB family identifier is: c3f47cf3.71b111cd.a4f2.00:01:80:31:7a:a7
  Comment supplied at packet creation is:
  Packet intended for the following targets:
   sanfran_hub [ local to this network ] tag: /vobs/tests
  The packet sequence number is 1
  ```

- List all packets in all of the local host's storage bays.

  multitool lspacket
  ```
  Packet is: c:\Program Files\Rational\ClearCase\var\shipping
  \ms_ship\incoming\packet1
  ...
  Packet is: c:\Program Files\Rational\ClearCase\var\shipping
  \ms_ship\incoming\packet2
  ```

- List all packets in a specific storage bay.

  multitool lspacket "c:\Program Files\Rational\ClearCase\var\shipping\to_boston"
  ```
  Packet is: c:\Program Files\Rational\ClearCase\var\shipping
  \to_boston\outgoing\packet1
  Packet type: Update
  ...
  ```

- List an update packet in long format.

  multitool lspacket –long /usr/tmp/packet1
  ```
  Packet is: /usr/tmp/packet1
  Packet type: Update
  VOB family identifier is: c3f47cf3.71b111cd.a4f2.00:01:80:31:7a:a7
  Comment supplied at packet creation is:
  Packet intended for the following targets:
   sanfran_hub [ local to this network ] * tag: /vobs/tests
  Originating replica is: sydney
  The following replicas are referenced by this packet:
   f3b1cd51.04b111d3.b2f0.00:c0:4f:96:17:d8
    first oplog id is 10
    incarnation is 06/29/95 12:18:09
   3f370590.04b211d3.b2f0.00:c0:4f:96:17:d8
    first oplog id is 0
    incarnation is 0
   8b354320.04c218k3.b5r0.00:c0:4f:99:27:f7
    first oplog id is 1
    incarnation is 07/21/95 11:45:20
  The major packet version is 2, the minor packet version is 0
  The packet sequence number is 1
  ```

SEE ALSO

mkreplica, **MultiSite Control Panel** (Windows), **syncreplica**, **shipping.conf** (UNIX)

lsreplica

Lists VOB replicas

APPLICABILITY

Product	Command type
ClearCase	cleartool subcommand
Attache	command
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

lsrep·lica [**–l·ong** | **–s·hort** | **–fmt** *format*]
[**–sib·lings**
| [**–sib·lings**] **–invob** *vob-selector*
| *replica-selector* ...
]

DESCRIPTION

This command lists information about all VOB-replica objects recorded in the VOB database of the current replica (except for deleted replicas). Other replicas may exist, but the packets containing their creation information have not yet been imported at the current replica.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

LISTING FORMAT. *Default:* Includes creation event information for each replica.

–l·ong

Includes each replica's creation information, master replica, mastership request setting, ownership information, and host. If the current replica is in the process of restoration, this option annotates the listings of other replicas from which restoration updates are required. (See the **restorereplica** reference page.)

–s·hort
Lists only replica names.

–fmt *format*
Lists information using the specified format string. See the **fmt_ccase** reference page for details on using this report-writing facility.

–sib·lings
Lists the VOB family members of the current replica, but does not list the current replica itself. This option is useful when you are writing scripts that process only sibling replicas.

SPECIFYING THE VOB FAMILY. *Default:* Lists VOB family members of the replica containing the current working directory.

–invob *vob-selector*
Lists the replicas of the specified VOB family. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

SPECIFYING THE REPLICA. *Default:* Lists all known replicas of the VOB family.

replica-selector ...
Restricts the listing to one or more replicas. Specify *replica-selector* in the form [**replica:**]*replica-name*[*@vob-selector*]

replica-name	Name of the replica
vob-selector	VOB family of the replica; can be omitted if the current working directory is within the VOB. Specify *vob-selector* in the form [**vob:**]*pname-in-vob* *pname-in-vob* — Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

- List the names of all replicas of the VOB containing the current working directory.

  multitool lsreplica –short
  ```
  bangalore
  boston_hub
  buenosaires
  sanfran_hub
  ```

- List the names of all siblings of the VOB containing the current working directory.

  multitool lsreplica –short –siblings
  ```
  bangalore
  buenosaires
  sanfran_hub
  ```

- Display a long listing of the current VOB's replicas.

  multitool lsreplica –long
  ```
  replica "bangalore"
   15-Aug-00.15:48:39 by Susan Goechs (susan.user@minuteman)
    replica type: unfiltered
    master replica: boston_hub@/vobs/dev
    request for mastership:enabled
    owner: susan
    group: user
    host: "ramohalli"
  replica "boston_hub"
   19-May-99.15:47:13 by Susan Goechs (susan.user@minuteman)
    replica type: unfiltered
    master replica: boston_hub@/vobs/dev
    request for mastership:enabled
    owner: susan
    group: user
    host: "minuteman"
  replica "buenosaires"
  ```

```
 15-Aug-00.15:48:44 by Susan Goechs (susan.user@minuteman)
  replica type: unfiltered
  master replica: boston_hub@/vobs/dev
  request for mastership:enabled
  owner: susan
  group: user
  host: "mardelplata"
replica "sanfran_hub"
 19-May-99.15:49:46 by Susan Goechs (susan.user@minuteman)
  replica type: unfiltered
  master replica: sanfran_hub@/vobs/dev
  request for mastership:enabled
  owner: susan
  group: user
  host: "goldengate"
```

- List all replicas of the VOB whose VOB-tag is **\doc**.

  multitool lsreplica –invob \doc

```
For VOB replica "\doc":
11-Mar.13:42        jcole               replica "boston_hub"
11-Mar.13:45        jcole               replica "sanfran_hub"
11-Mar.13:48        jcole               replica "tokyo"
```

- List the name, master replica, and replica host of all replicas in the VOB family **/vobs/doc**.

  cmd-context **lsreplica –fmt **
  **"Name: %n\n\tMaster replica: %[master]p\n\tReplica host: %[replica_host]p\n" **
  -invob /vobs/doc

```
Name: boston_hub
   Master replica: boston@/vobs/doc
   Replica host: minuteman
Name: sanfran_hub
   Master replica: sanfran_hub@/vobs/doc
   Replica host: goldengate
Name: tokyo
   Master replica: sanfran_hub@/vobs/doc
   Replica host: shinjuku
```

SEE ALSO

mkreplica

mkorder

Creates a shipping order for use by the store-and-forward facility

APPLICABILITY

Product	Command type
MultiSite	MultiSite command

Platform
UNIX
Windows

SYNOPSIS

mkorder –dat·a *packet-pname* [**–scl·ass** *storage-class-name*]
[**–pex·pire** *date-time*] [**–not·ify** *e-mail-address*]
[**–c·omment** *comment* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
[**–shi·p –cop·y** | **–fsh·ip** [**–cop·y**] | **–out** *order-pname*] *destination ...*

This command is located in *ccase-home-dir***/etc** on UNIX and *ccase-home-dir***\bin** on Windows.

DESCRIPTION

This command creates a shipping order file for an existing packet or any other file. The shipping order is used by the **shipping_server** command to send the packet to one or more destinations.

mkorder submits to the store-and-forward facility a packet that was created with **mkreplica –out** or **syncreplica –out**. You can also use **mkorder** to resubmit store-and-forward packets whose shipping orders have expired, and to transfer other files among sites.

A shipping order must be located in the same directory as its associated packet or file.

NOTE: The store-and-forward facility deletes a packet after delivering it successfully (except when the destination is the local host). If you use this command to process a file that must be preserved at your site even after delivery to another site, you must specify the **–copy** option.

RESTRICTIONS

None.

OPTIONS AND ARGUMENTS

SPECIFYING THE PACKET FILE. *Default:* None.

–dat·a *packet-pname*
The pathname of the packet or file.

NOTE: If *packet-pname* contains a colon character (**:**), **mkorder** changes the colon to a period character (**.**) during processing. This allows packets to be delivered to Windows machines, which do not allow colons within file names.

SPECIFYING WHERE TO PLACE THE SHIPPING ORDER. *Default:* Creates a shipping order in the directory where the *packet-pname* file is located.

–scl·ass *class-name*
Specifies the storage class of the packet and shipping order. If you also use **–ship** or **–fship**, **mkorder** looks up the storage class in the store-and-forward configuration settings (on Windows, in the **MultiSite Control Panel**; on UNIX, in the file *ccase-home-dir***/config/services/shipping.conf**) to determine the location of the storage bay to use.

If you omit this option but use **–ship** or **–fship**, **mkorder** places the shipping order in the storage bay location specified for the **–default** class in the MultiSite Control Panel or the **shipping.conf** file.

–shi·p –cop·y
–fsh·ip [**–cop·y**]
Creates a shipping order for the *packet-pname* file. Using **–fship** (force ship) invokes **shipping_server** to send the packet. Using **–ship** places the shipping order in a storage bay. To send the packet, run **shipping_server** or set up invocations of **sync_export_list –poll** with the **schedule** command. (See the **schedule** reference page in the *Command Reference*.)

–copy is required with **–ship**, and optional with **–fship**:

- With **–copy**, **mkorder** copies the *packet-pname* file to one of the store-and-forward facility's storage bays, and places the shipping order in the bay. The copy is deleted after it is delivered successfully to all the destinations specified in the shipping order.
- Without **–copy**, **mkorder** does not copy *packet-pname*; **mkorder** places the shipping order in the directory where the file is located. *packet-pname* is deleted after it is delivered successfully to all the destinations specified in the shipping order.

–out *order-pname*

Places the shipping order in the specified file instead of in a storage bay. An error occurs if the file already exists.

HANDLING PACKET-DELIVERY FAILURES. *Default:* If a packet cannot be delivered, it is sent through the store-and-forward facility back to the administrator at the site of the originating replica. A mail message is sent to the store-and-forward administrator. This occurs after repeated attempts to deliver the packet have failed, and the allotted time has expired; it can also occur when the destination host is unknown or a data file does not exist. The store-and-forward configuration settings specify the expiration period and the e-mail address of the administrator.

–pex·pire *date-time*

Specifies the time at which the store-and-forward facility stops attempting to deliver the packet and generates a failure mail message instead.

UNIX: This option overrides the storage class's **EXPIRATION** specification in the store-and-forward configuration file. See the **shipping.conf** reference page for a discussion of this specification and of delivery retries in general.

Windows: This option overrides the storage class's Packet Expiration specification in the MultiSite Control Panel. See the **MultiSite Control Panel** reference page for a discussion of this specification, and of delivery retries in general.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**
where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** | **yesterday** | **Sunday** | ... | **Saturday** | **Sun** | ... | **Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... | **December** | **Jan** |... | **Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–not·ify *e-mail-address*
The delivery-failure message is sent to the specified e-mail address.

If a failure occurs on a Windows host that does not have e-mail notification enabled, a message appears in the Windows Event Viewer. The message includes the *e-mail-address* value specified with this option and a note requesting that this user be informed of the status of the operation. For information about enabling e-mail notification, see the **MultiSite Control Panel** reference page.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page.

–c·omment *comment* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with one of the MultiSite comment options.

SPECIFYING THE DESTINATION. *Default:* None.

destination ...
One or more host names (which must be usable by hosts in different domains) or IP addresses. When sending a MultiSite packet, you must specify the host where the replica actually resides or is to be created.

EXAMPLES

- Create a shipping order for file **p1**, which is located in the default storage bay. Store the shipping order in the same storage bay as **p1**, and specify that the file is to be sent to host **goldengate**. The lines are broken for readability. You must enter the command on a single physical line.

  mkorder –data "c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\p1" –out "c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\p1_order" goldengate

  ```
  Shipping order "c:\Program Files\Rational\ClearCase\var
  \shipping\ms_ship\p1_order" generated.
  ```

- Create a shipping order in the default storage bay for a specified file that is to be delivered to host **goldengate**. Specify that **admin** must be notified if the file is not delivered successfully.

  /usr/atria/etc/mkorder –data /usr/tmp/to_goldengate –ship –copy –notify admin goldengate
  ```
  Shipping order "/var/adm/atria/shipping/ms_ship/sh_o_to_goldengate"
  generated.
  ```

- Create a shipping order for the same file, but place it in the storage bay for a particular storage class. Attempt immediate delivery (**–fship**), and allow delivery attempts to continue until the beginning of May 18.

  mkorder –data c:\tmp\to_goldengate –fship –copy –sclass ClassA –pexpire 18-May goldengate
  ```
  Shipping order "c:\tmp\sclass\ClassA\sh_o_to_goldengate" generated.
  Attempting to forward/deliver generated packets...
  -- Forwarded/delivered packet c:\tmp\sclass\ClassA\sh_o_to_goldengate
  ```

FILES

ccase-home-dir/**config/services/shipping.conf**

SEE ALSO

mkreplica, **MultiSite Control Panel**, **shipping.conf**, **shipping_server**, **syncreplica**
Chapter 10, *Troubleshooting MultiSite Operations*

mkreplica

Creates a VOB replica

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

- Duplicate an existing VOB replica, generating a replica-creation packet:

  mkrep·lica –exp·ort –wor·kdir *temp-dir-pname* [**–max·size** *size*]
  [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
  { { **–sh·ip** | **–fshi·p** } [**–scl·ass** *storage-class*] [**–pex·pire** *date-time*] [**–not·ify** *e-mail-addr*]
  | **–tape** *raw-device-pname*
  | **–out** *packet-file-pname*
  }
  hostname:*replica-selector* ...

  NOTE: The **–tape** option is valid only on UNIX.

- Use a replica-creation packet to create a new VOB replica:

  mkrep·lica –imp·ort –wor·kdir *temp-dir-pname* **–tag** *vob-tag*
  { **–vob** *vob-stg-pname* [**–hos·t** *hostname* **–hpa·th** *host-stg-pname* **–gpa·th** *global-stg-pname*]
  | **–stgloc** { *stgloc-name* | **–auto** } }
  { **–pre·serve** | **–npr·eserve** }
  [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
  [**–tco·mment** *tag-comment*] [**–nca·exported**]
  [**–reg·ion** *region-name*] [**–opt·ions** *mount-options*]
  [**–pub·lic** [**–pas·sword** *tag-registry-password*]] [**–ign·oreprot**]
  [**–poo·ltalk**] [**–vre·plica** *replica-name*]
  { **–tap·e** *raw-device-pname* | *packet-file-pname* [*search-dir-pname* ...] }

  NOTE: The **–ncaexported** and **–tape** options are valid only on UNIX.

DESCRIPTION

The creation of a new VOB replica is a two-phase process. Both phases require you to enter a **mkreplica** command:

1. The **mkreplica –export** command duplicates the contents of the current VOB replica (the originating replica). This generates a single logical replica-creation packet for transmission to one or more other sites. As described in *REPLICA-CREATION PACKETS* on page 251, it may be divided into multiple physical packets. (If you use **–fship** or **–ship**, **mkreplica** also generates a shipping order file for each physical packet.)

   This command also creates a new replica object in the VOB database.

   The VOB is locked for the entire length of time the **mkreplica –export** command runs.

   NOTE: Creating multiple replicas in one **mkreplica –export** command is more efficient than using multiple **mkreplica –export** commands.

2. At another site, a **mkreplica –import** command uses the replica-creation packet to create a new VOB replica. The user who enters this command becomes the VOB owner of the new replica.

When a VOB is first replicated, creating a second replica, the VOB's oplog (operation log) is enabled. All ClearCase and MultiSite operations to be replicated are recorded in the oplog. Logging of operations continues until all but one of the VOB's replicas are deleted. Note that creation of additional replicas is recorded in oplog entries. Existing replicas learn about a newly created replica through the standard synchronization mechanism. (See the **syncreplica** reference page.)

NOTE: Before entering a **mkreplica –export** command, make sure MultiSite licenses are installed at the original site. After you enable replication in the original VOB, developers cannot access the VOB without a MultiSite license (in addition to a ClearCase license).

OWNERSHIP PRESERVATION

When you enter a **mkreplica –import** command, you must choose whether to make the new replica ownership-preserving or non-ownership-preserving. In either case, the user who enters the **mkreplica –import** command becomes the owner of the new VOB replica. Ownership preservation affects only element ownership and permissions. For more information on ownership preservation, see *Element Ownership and Ownership Preservation* on page 4.

Restrictions:

- Creating an ownership-preserving replica is appropriate only if its site supports the same user and group accounts as the originating site. On Windows, therefore, if replicas in a VOB family are not all in the same Windows domain, the entire set of replicas cannot be ownership-preserving. However, you can maintain ownership preservation on the subset of replicas in the same domain.

- **Windows**: The primary group of the user who enters the **mkreplica –import** command must be the same as the originating replica's group assignment.
- **UNIX**: The user who enters the **mkreplica –import** command must belong to all the groups on the originating replica's group list.

NOTE: We recommend that you run **syncreplica –export** immediately after creating a new replica with **mkreplica –import –preserve**, to inform other replicas in the VOB family that the new replica is ownership-preserving.

REPLICA-CREATION PACKETS

Each invocation of **mkreplica –export** creates a single logical replica-creation packet. (This is true even if you create several new replicas with one **mkreplica** command.) Each packet carries one or more replica specifications, each of which indicates the host on which a new replica is to be created, along with the new replica's name.

The **–maxsize** option divides the single logical packet into multiple physical packets to conform with limitations of the transfer medium.

Cleaning Up Used Packets

Replica-creation packets are not deleted after import. The VOB owner at the new replica site must delete replica-creation packets after importing them with **mkreplica –import**.

REPLICATION OF VOBS LINKED TO ADMINISTRATIVE VOBS

If the VOB you are replicating is linked to an administrative VOB, **mkreplica –export** prints a reminder that you must replicate all administrative VOBs in the hierarchy above the VOB you are replicating. The output lists the administrative VOBs. The command does not check whether these administrative VOBs are replicated, so you can ignore the message if you have already replicated them.

RESTRICTIONS

Identities: For **mkreplica –export**, you must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB.

Mastership: No mastership restrictions.

Other:

- You must execute **mkreplica –export** on the host where the VOB storage directory resides.
- You cannot replicate a VOB to a host running an earlier major version of MultiSite. (However, you can replicate a VOB to a host running a later major version of MultiSite.)

OPTIONS AND ARGUMENTS — EXPORT PHASE

The following sections describe the options and arguments for use with **mkreplica –export**.

SPECIFYING TEMPORARY WORKSPACE. *Default:* None.

–wor·kdir *temp-dir-pname*
: A directory for use by **mkreplica** as a temporary workspace; it is deleted when **mkreplica** finishes. This directory must not already exist. You must specify a location in a disk partition that has enough free space (at least the size of the VOB database directory plus its source pools; use **cleartool space** to display VOB disk space use).

SPECIFYING THE REPLICA-CREATION PACKET SIZE. *Default:* When you do not specify **–maxsize**, the default packet size depends on the shipping method you use:

- Packets created with **–ship** or **–fship** are no larger than the maximum packet size specified in the **shipping.conf** file (UNIX) or the **MultiSite Control Panel** (Windows).
- Packets created with **–out** are no larger than 2 GB.
- Packets created with **–tape** have no default size limit.

The **mkreplica** command fails if it tries to create a packet larger than the size supported by your system or by the tape.

–max·size *size*
: The maximum size for a physical packet, expressed as a number followed by a single letter; for example:

500k	500 kilobytes
20m	20 megabytes
1.5g	1.5 gigabytes

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–cqe**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment-string* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
: Overrides the default with the specified comment option.

DISPOSITION OF THE REPLICA-CREATION PACKET. *Default:* None. You must specify how the replica-creation packet created by **mkreplica –export** is to be stored and/or transmitted to other sites.

–shi·p
–fsh·ip
: Stores the replica-creation packet in one or more files in a store-and-forward storage bay.

A separate shipping order file accompanies each physical packet, indicating how and where it is to be delivered.

–fship (force ship) invokes **shipping_server** to send the replica-creation packet. **–ship** places the packet in a storage bay. To send the packet, invoke **shipping_server** or set up invocations of **sync_export_list –poll** with the **schedule** command. (See the **schedule** reference page in the *Command Reference*.)

NOTE: The disk partition where the storage bay is located (on the sending host and the receiving host) must have available space equal to or greater than the size of the VOB database and source pools.

–scl·ass *class-name*
Specifies the storage class of the packet and shipping order. **mkreplica** looks up the storage class in the store-and-forward configuration settings (on Windows, in the **MultiSite Control Panel**; on UNIX, in the file *ccase-home-dir***/config/services/shipping.conf**) to determine the location of the storage bay to use.

If you omit this option, **mkreplica** places the packet in the storage bay location specified for the **–default** class.

–tap·e *raw-device-pname* (UNIX)
Writes the replica-creation packets to the specified tape device, which must be local to the VOB server host. You are prompted to load a separate tape for each physical packet. Use the **–maxsize** option to ensure that **syncreplica** does not exceed the capacity of the tapes you are using. Only one physical packet can be placed on each tape, regardless of packet size.

–out *packet-file-pname*
Places the first physical replica-creation packet in file *packet-file-pname*. Additional packets are placed in files named *packet-file-pname_2*, *packet-file-pname_3*, and so on.

The replica-creation packets are not delivered automatically; use an appropriate mechanism (for example, electronic mail, **ftp**, or postal service) to deliver them.

You can create a packet using **–out**, and subsequently deliver it using the store-and-forward facility. See the **mkorder** reference page for details.

HANDLING PACKET-DELIVERY FAILURES. *Default:* If a packet cannot be delivered, it is sent through the store-and-forward facility back to the administrator at the site of the originating replica. A mail message is sent to the store-and-forward administrator. This occurs after repeated attempts to deliver the packet have all failed, and the allotted time has expired; it can also occur when the destination host is unknown or a data file does not exist. The store-and-forward configuration settings specify the expiration period and the e-mail address of the administrator).

–pex·pire *date-time*

Specifies the time at which the store-and-forward facility stops trying to deliver the packet and generates a failure mail message instead.

UNIX: This option overrides the storage class's **EXPIRATION** specification in the store-and-forward configuration file. See the **shipping.conf** reference page for a discussion of this specification and of delivery retries in general.

Windows: This option overrides the storage class's Packet Expiration specification in the MultiSite Control Panel. See the **MultiSite Control Panel** reference page for a discussion of this specification and of delivery retries in general.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**
where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** | **yesterday** | **Sunday** | ... | **Saturday** | **Sun** | ... | **Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–not·ify *e-mail-address*

The delivery-failure message is sent to the specified e-mail address.

If a failure occurs on a Windows host that does not have e-mail notification enabled, a message appears in the Windows Event Viewer. The message includes the *e-mail-address* value specified with this option and a note requesting that this user be informed of the status of the operation. For information about enabling e-mail notification, see the **MultiSite Control Panel** reference page.

REPLICA SPECIFICATIONS. *Default:* None.

hostname:replica-selector...

One or more arguments, each of which indicates one new replica to be created from this packet at another site.

hostname

Names the machine where the new replica's storage directory will be created. *hostname* must be usable by hosts in different domains. It is used by the ClearCase store-and-forward mechanism to determine how to route update packets to the replica. However, keep this information accurate even if your site does not use store-and-forward. (See the **chreplica** reference page.)

hostname can be either the IP address of the host or the computer name, for example, **minuteman**. You may have to append an IP domain name, for example, **minuteman.purpledoc.com**.

On UNIX, use the **uname –n** command to display the computer name. On Windows NT, the computer name is displayed in the **Network Settings** dialog box, which is accessible from the **Network** icon in the Control Panel. On Windows 2000, the computer name is displayed on the **Network Identification** tab in the **System Properties** dialog box, which is accessible from the **System** icon in the Control Panel.

Specify *replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*]

replica-name Name of the replica

You must compose the name according to these rules:

- It must contain only letters, digits, and the special characters underscore (_), period (.), and hyphen (-). A hyphen cannot be used as the first character of a name.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)
- It must not be one of the special names " **.** ", " **..** ", or " **...** ".

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

OPTIONS AND ARGUMENTS — IMPORT PHASE

The following sections describe the options and arguments for use with **mkreplica –import**.

SPECIFYING TEMPORARY WORKSPACE. *Default:* None.

–wor·kdir *temp-dir-pname*
A directory for use by **mkreplica** as a temporary workspace; it is deleted when **mkreplica** finishes. This directory must not already exist. Make sure to specify a location in a disk partition that has enough free space. (See the description of **–workdir** in *OPTIONS AND ARGUMENTS — EXPORT PHASE*.)

SPECIFYING VOB-CREATION PARAMETERS. *Default:* Because **mkreplica –import** executes a **cleartool mkvob** command, you can use many of the options used with **mkvob**. The **–tag** option is required, and one of **–vob** or **–stgloc** is required. See the **mkvob** reference page in the *Command Reference* for detailed descriptions of these options.

–tag *vob-tag*
The VOB-tag (mount point) of the new VOB replica.

–vob *vob-stg-pname*
Location for the storage directory of the new VOB replica. On Windows, *vob-stg-pname* must be a UNC name.

–hos·t *hostname* | **–hpa·th** *host-stg-pname* | **–gpa·th** *global-stg-pname*
Sets the new VOB replica's registry information. In most cases, **mkreplica** derives this information from the *vob-storage-pname* argument, but if your network topology is unusual or the network interface is not standard, you may have to use these options. If you have to use these options when creating a new VOB at the site, you have to use them when importing a replica-creation packet.

–stgloc { *stgloc-name* | **–auto** }
Specifies the name of a storage location for the new replica's VOB storage directory. *stgloc-name* must be located on the same host on which you invoke **mkreplica**, and it must be one of the registered storage locations. To list registered locations, use **cleartool lsstgloc**. With **–auto**, **mkreplica** selects a location automatically.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Standard comment options.

–tco·mment *tag-comment*
A comment string to be included in the VOB-tag registry entry for the new replica.

–nca·exported (UNIX)
Marks the new VOB replica for NFS export.

–reg·ion *region-name*
Specifies an alternate registry region for the new replica's VOB-tag.

–opt·ions *mount-options*
Mount options for the new VOB replica.

–pub·lic [**–pas·sword** *tag-registry-password*]
Creates a public VOB-tag for the new replica.

PROTECTION FAILURES ON CONTAINERS. *Default:* During import, if any data containers have a group that is not the primary group of the VOB, a failure occurs when **mkreplica** tries to set the protection of those containers. The import fails if protection failures occur.

–ign·oreprot
Completes the import even if protection failures occur. **mkreplica** prints a warning that the protection problems may make the replica unusable. You must run **checkvob** to find and fix any problems after creating a replica with this option.

NOTE: Instead of using this option, you can add the nonprimary groups to the group list of the user importing the packet.

OWNERSHIP PRESERVATION. *Default:* None.

–pre·serve
Creates a replica that is ownership-preserving. The user who enters the **mkreplica –import** command becomes the owner of the new VOB, and ownership and permissions are preserved for all the elements in the new VOB.

–npr·eserve
Creates a replica that is not ownership-preserving. The user who enters the **mkreplica –import** command becomes the owner of the new VOB and of all the elements in the new VOB.

POOL CREATION FOR THE NEW REPLICA. *Default:* The new replica is created with the same set of storage pools as the originating replica, and the assignments of elements to pools are preserved. The new replica's storage pools are created within its storage directory, even if some of the originating replica's pools are remote; the new pools have the default scrubbing parameters.

–poo·ltalk
Prompts you to specify locations and scrubbing specifications for the new replica's storage pools.

NAME OF VOB REPLICA. *Default:* If the replica-creation packet includes one replica specification, you are prompted to confirm the replica name. If the packet includes multiple replica specifications, you are prompted to select one of the replica names.

–vre·plica *replica-name*
Specifies the replica name, bypassing the prompt step.

SPECIFYING THE LOCATION OF THE REPLICA-CREATION PACKET. *Default:* None.

–tap·e *raw-device-pname* (UNIX)
Reads a replica-creation packet from the specified tape device, which must be local to the host on which you enter the **mkreplica –import** command. Before entering the command, place the tape in the tape drive. If a logical packet spans multiple tapes, you can start with any of them in the drive. You are prompted to switch tapes.

packet-file-pname [*search-dir-pname* ...]
Specifies a pathname of a replica-creation packet created by **mkreplica –export**. For a logical packet that spans multiple disk files, **mkreplica** scans the directory containing *packet-file-pname* for related physical packets.

If you also specify one or more *search-dir-pname* arguments, **mkreplica** searches for additional packets in these directories.

EXAMPLES

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

Exports

- Generate a replica-creation packet, which will be used at remote host **goldengate** to create a new replica named **sanfran_hub**. Place the packet in a file in directory **/tmp**.

  multitool mkreplica –export –workdir /tmp/ms_workdir –c "make a new replica for sanfran_hub" –out /tmp/sanfran_hub_packet goldengate:sanfran_hub

  ```
  Generating replica creation packet /tmp/sanfran_hub_packet
  Dumping database...
  ...
  Dumper done.
  ```

- Generate a replica-creation packet and place it in a storage bay.

multitool mkreplica –export –c "make a new replica for sanfran_hub" –workdir /tmp/ms_workdir –ship goldengate:sanfran_hub

```
Generating replica creation packet
/var/adm/atria/shipping/ms_ship/outgoing/repl_boston_hub_18-May-99.15:50:00_1
 - shipping order file is
/var/adm/atria/shipping/ms_ship/outgoing/sh_o_repl_boston_hub_18-May-99.15:50:
00_1
Dumping database...
...
Dumper done.
```

- Generate a replica-creation packet that can be used to create two new replicas, **bangalore** and **buenosaires**. Ship the packet to its destinations immediately, using store-and-forward.

multitool mkreplica –export –workdir /tmp/ms_workdir –nc –fship ramohalli:bangalore mardelplata:buenosaires

```
Generating replica creation packet
/usr/atria/shipping/ms_ship/outgoing/repl_boston_hub_15-Aug-00.14.26.17_6011_1
- shipping order file is
/usr/atria/shipping/ms_ship/outgoing/sh_o_repl_boston_hub_15-Aug-00.14.26.17_6
011_1
Dumping database...
...
Dumper done.
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/usr/atria/shipping/ms_ship/outgoing/repl_boston_hub_15-Aug-00.14.26.17_6011_1
```

Imports

- Using a packet file in **/tmp**, create the storage directory for replica **sanfran_hub**. Make the replica ownership-preserving, and immediately after creating the new replica, run **syncreplica –export** to update the other replicas in the VOB family.

  multitool mkreplica –import –workdir /tmp/ms_workdir
  –tag /vobs/dev –vob /net/goldengate/vobstg/dev.vbs
  –preserve –c "create sanfran_hub replica" /tmp/sanfran_hub_packet

  ```
  The packet can only be used to create replica "sanfran_hub"
   - VOB family is c3f47cf3.71b111cd.a4f2.00:01:80:31:7a:a7
   - replica OID is 0c39c3b8.727b11cd.abb5.00:01:80:31:7a:a7
  Should I create this replica? [no] yes
  Processing packet /tmp/sanfran_hub_packet...
  Loading database...
  ...
  Loader done.
  Registering VOB mount tag "/vobs/dev"...
  VOB replica successfully created.
  Host-local path: goldengate:/vobstg/dev.vbs
  Global path: /net/goldengate/vobstg/dev.vbs
  VOB ownership:
   owner ...
   group ...
  ```

  multitool syncreplica –export –c "ownership-preserving" –fship boston_hub bangalore buenosaires

  ```
  ...
  ```

- Similar to preceding example, but create the replica as a public VOB and non-ownership-preserving. Specify the VOB-tag password and mount options on the command line.

  multitool mkreplica –import –workdir /tmp/ms_workdir
  –tag /vobs/dev –vob /net/goldengate/vobstg/dev.vbs
  –npreserve –c "create sanfran_hub replica" –options rw,soft
  –public –password xxxxxx –vreplica sanfran_hub /tmp/sanfran_hub_packet

  ```
  Processing packet /tmp/sanfran_hub_packet...
  ...
  Registering VOB mount tag "/vobs/dev"...
  VOB replica successfully created.
  ...
  ```

- Create the storage directory for a new replica, using a packet that was generated by existing replica **boston_hub** and sent through store-and-forward. Specify storage pool parameters for the new replica.

  multitool mkreplica –import –workdir c:\tmp\workdir –tag \dev –vob \\ramohalli\vobs\dev.vbs –npreserve –c "create bangalore replica" –pooltalk –vreplica bangalore "c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\incoming\repl_boston_hub_15-Aug-00.14.26.17_6011_1

```
Processing packet c:\Program
Files\Rational\ClearCase\var\shipping\ms_ship\incoming\repl_boston_hub_15-
Aug-00.14.26.17_6011_1
The initial storage pools that will be used in the replica are:
 source pool sdft
 derived pool ddft
 cleartext pool cdft
Configuration for pool "sdft" (source pool):
Full pathname of directory to which pool "sdft"
should be linked (none = not linked)? [none] <RETURN>

Configuration for pool "ddft" (derived pool):
Full pathname of directory to which pool "ddft"
should be linked (none = not linked)? [none] <RETURN>

Maximum size (in Kbytes) for the storage directory of pool "ddft"
 (0 = no maximum)? [0] <RETURN>
Space (in Kbytes) to reclaim from pool "ddft"
during scrubbing (0 = none)? [0] <RETURN>

Minimum age (in hours) of objects to scrub from pool "ddft"
(0 = none)? [0] 12
Command to invoke if scrubbing does not reduce pool "ddft"
below maximum size (none = no command)? [none] <RETURN>
Comment for pool "ddft" (none = none)? [none] <RETURN>
 . . . (accept defaults for cleartext pool, cdft)

                           Max. Reclaim   Min. Link To
Pool Name   Kind           Size   Size    Age Directory
---------   ----           ----   ----    --- ---------
sdft        source pool    n/a    n/a     n/a
ddft        derived pool   0K     0K      12
cdft        cleartext pool 0K     0K      96

Is this the correct configuration for the pools (yes/no/abort)? [no] yes
...
Registering VOB mount tag "\dev"...
...
```

SEE ALSO

chmaster, **chreplica**, **lspacket**, **lsreplica**, **mkorder**, **MultiSite Control Panel**, **shipping.conf**, **syncreplica**, **mkvob** (in the *Command Reference*)
Chapter 10, *Troubleshooting MultiSite Operations*

MultiSite Control Panel

Configures store-and-forward facility

APPLICABILITY

Product	Command type
MultiSite	Administrative tool

Platform
Windows

SYNOPSIS

%SystemRoot%\System32\ms.cpl
To open the MultiSite Control Panel, double-click the MultiSite icon in Control Panel.

DESCRIPTION

The MultiSite Control Panel controls operation of the MultiSite store-and-forward facility on each host. MultiSite installation creates registry keys in which all these entries are defined. In some cases, the corresponding store-and-forward operation fails if a parameter is not defined, and in other cases there is a hard-coded default.

The MultiSite Control Panel provides controls for setting the configuration parameters described in the following sections.

MAXIMUM PACKET SIZE

Controls the splitting of individual logical packets into multiple physical packets. This value specifies the maximum size for a physical packet file. Limiting the size of physical packets can improve the reliability of packet delivery in some networks. To specify no limit, use 0 (zero).

This value is used by the following commands (unless you also specify **–maxsize**):

- **mkreplica –fship**
- **mkreplica –ship**
- **syncreplica –fship**
- **syncreplica –ship**
- **sync_export_list**

When you invoke **mkreplica** or **syncreplica** with **–out**, this value is not used, and you must use **–maxsize** to limit the packet size.

Default: 2097151KB

ADMINISTRATOR E-MAIL

Specifies the electronic mail address of the user to be notified when any of these events occur:

- A packet (on the local host) that has expired is returned to its sending host.
- A packet that was not delivered to its next hop is returned to its sending host.
- **syncreplica –import** finds a replica-creation packet.

To enable e-mail notification:

1. Make sure the **SMTP Host** box in the ClearCase Control Panel specifies a valid host. (This box is located on the **Options** tab.)
2. Enter an e-mail address in the **Administrator Email** box in the MultiSite Control Panel. You can specify only one address.

Default: None.

STORAGE CLASSES

Storage Class Name

Specifies the name of a storage class. You can associate values for packet expiration, the storage bay, the return bay, and the receipt handler with each storage class.

Default: ClearCase MultiSite installation sets up a default storage class (**–default**) with predefined values. The **–default** class is used when you invoke the **mkorder**, **mkreplica**, **syncreplica**, or **sync_export_list** command with the **–fship** or **–ship** option and do not specify a storage class. You can change the values associated with the **–default** class.

NOTE: The name for the ClearQuest MultiSite storage class must be **cq_default**.

Packet Expiration

Specifies the expiration period (in days) for shipping orders generated in the associated storage class. This period begins at the time the shipping order is generated. If a packet cannot be delivered to all its specified destinations in the specified number of days, the packet is returned to the original sending host and a message is sent to the address specified in the **Administrator Email** box. If e-mail notification is not enabled, a message is written to the Windows Event Viewer.

A value of **0** (zero) specifies no expiration; delivery is reattempted indefinitely.

This setting is overridden by the **–pexpire** option to **syncreplica** or **mkreplica**.

The **shipping_server** program does not retry delivery of packets. The Packet Expiration specification is useful only if you set up a host to periodically attempt delivery of any undelivered packets. To set up redelivery attempts, use the **schedule** command to invoke **sync_export_list –poll**, which invokes **shipping_server –poll**. For more information, see the **schedule** reference page in the *Command Reference*.

Default: When the **Use Default Expiration** check box is selected, the storage class uses the packet expiration value associated with the **–default** class. (This value is not shown in the **Packet Expiration** box; you must display the **–default** class to determine the value.) When MultiSite is installed for the first time, the Packet Expiration value for the **–default** class is set to 14 days.

Storage Bay Path

Defines the location of a storage bay, a directory that holds the outgoing and incoming update packets and shipping orders of a particular storage class.

Packets placed in a storage bay on an NTFS file system inherit the Windows ACL on the storage bay. Define ACLs on the storage and return bays to enable successful execution of MultiSite commands to process the packets, and to guard against unauthorized access. (If you use the **schedule** command to invoke **sync_export_list –poll** on **shipping_server**, the group **ClearCase** must have read and write permissions on all storage directories.) Packets stored on FAT file systems have no protections.

Before using the store-and-forward facility, make sure that the disk partition where the *ccase-home-dir***var\shipping** directory is created has sufficient free space for anticipated replica-creation and update packets. The amount of available space on the disk partition where the shipping and return bays are located must be at least twice as big as the largest packet that will be stored in the bays. This is because there may be two copies of the same packet in the bay at one time: one on its way to another destination, and another waiting to be applied to the replica on the host.

Default: When MultiSite is installed for the first time, the storage bay associated with the **–default** storage class is *ccase-home-dir***var\shipping\ms_ship**. This bay contains subdirectories named **incoming** and **outgoing**, which hold incoming and outgoing packets. Shipping operations look for packets in these subdirectories.

NOTE: When you create a new storage class, the storage bay and return bay that you specify are created, along with the **incoming** and **outgoing** directories within the bays.

Return Bay Path

Defines a return bay (directory) to hold incoming or outgoing packets in the process of being returned to their origin because they could not be delivered to all specified destinations.

Packets placed in a return bay inherit the ACL on the directory.

Default: When MultiSite is installed for the first time, the return bay associated with the **–default** storage class is *ccase-home-dir***var\shipping\ms_rtn**. This bay contains subdirectories named **incoming** and **outgoing**, which hold incoming and outgoing packets. Shipping operations look for packets in these subdirectories.

Receipt Handler Path

Specifies a batch file or program for the **shipping_server** to run when a packet is received for the storage class. You can use this instead of scheduling executions of **sync_receive**. By default, no file is specified. We recommend that you specify *ccase-home-dir***config\scheduler\tasks\sync_receive** in the **Receipt Handler Path** box.

For each packet that is received, **shipping_server** does the following:

1. Reads the entries in the MultiSite Control Panel to find the appropriate **Receipt Handler** value for the packet.
   - If the packet is associated with a storage class and there is a **Receipt Handler** value for that storage class, **shipping_server** uses the specified batch file or program
   - If the packet is not associated with a storage class and there is a **Receipt Handler** value for the **–default** storage class, **shipping_server** uses the batch file or program specified for **–default**
2. Invokes the receipt handler, as follows:

   script-pname [**–d·ata** *packet-file-pname*] [**–a·ctual** *shipping-order-pname*] [**–s·class** *storage-class*] **–o·rigin** *hostname*

   where

script-pname	Script specified in the **RECEIPT-HANDLER** entry.
–d·ata *packet-file-pname*	Location of the packet. This parameter is used only when the packet is destined for this host.
–a·ctual *shipping-order-pname*	Location of the shipping order. This parameter is used only when the packet is destined for another host.
–s·class *storage-class*	Storage class associated with the packet. This parameter is used only if the packet was associated with a storage class when it was created.
–o·rigin *hostname*	Host name of the machine from which the packet was first sent.

   NOTE: If a packet is destined for both the local host and another host, both the **–data** and **–actual** parameters are used. The packet is imported at the replica on the host, then forwarded to its next destination.

Default: None.

ROUTING INFORMATION

The Routing Information settings control the network routing of packets.

Next Routing Hop

The host specified here is the next destination for packets whose final destination is any of the host names specified in the **Destination Hostnames** list. This host is responsible for delivery of the packet to its destinations. You can specify a host using either its host name (which must be usable by hosts in different domains) or its numeric IP address.

Default: None.

Destination Host Names

Packets destined for any host listed in this field are sent to the host specified in the **Next Routing Hop** box. The value **–default** as the **Destination Hostname** accommodates all hosts that are not associated with a routing hop. You can specify a host using either its host name (which must be usable by hosts in different domains) or its numeric IP address.

Default: None.

multitool

MultiSite user-level commands

APPLICABILITY

Product	Command type
MultiSite	MultiSite command

Platform
UNIX
Windows

SYNOPSIS

- Single-command mode:

  multitool *subcommand* [*options/args*]

- Interactive mode:

  multitool [**–e**]

  `multitool>` *subcommand* `[` *options/args* `]`
  `. . .`

  `multitool>` **quit**

- Status mode:

  multitool –status

  `multitool 1>` *subcommand* `[` *options/args* `]`
  `. . .`

  `multitool 5>` **quit**

- Display version information for **multitool** (and on UNIX, MultiSite):

  multitool –ver·sion

- Display version information for **multitool** and the libraries used by **multitool** (and on UNIX, MultiSite):

  multitool –VerAll

DESCRIPTION

multitool is the principal program in MultiSite. Typically, you also use MultiSite extensions incorporated into **cleartool** subcommands in ClearCase.

The different **multitool** subcommands are described in *multitool Subcommands* on page 55.

USING INTERACTIVE MODE AND STATUS MODE

With **–e**, **multitool** enters interactive mode. It exits if an error is returned by a command.

With **–status**, **multitool** enters interactive mode and returns the status (**0** or **1**) of each **multitool** subcommand executed.

If you exit **multitool** by entering a **quit** command in interactive mode, the exit status is 0. The exit status from single-command mode depends on whether the command succeeded (zero exit status) or generated an error message (nonzero exit status).

SPECIFYING OBJECTS WITH OBJECT SELECTORS

In **multitool** commands, you specify non-file-system VOB objects (types, pools, hyperlinks, and replicas) with object selectors.

Object selectors identify non-file-system VOB objects with a single string:

[*prefix***:**]*name*[**@***vob-selector*]

where

prefix

The kind of object. The *prefix* is optional if the context of the command implies the kind of object. For example,

multitool lsreplica replica:buenosaires

is equivalent to

multitool lsreplica buenosaires

If a context does not imply any particular kind of object, **multitool** assumes that a *name* argument with no prefix is a pathname. For example, the command **multitool describe buenosaires** describes a file-system object named **buenosaires**, while **multitool describe replica:buenosaires** describes the **buenosaires** replica object.

name

The name of the object. See the *Object Names* section for the rules about composing names.

vob-selector

VOB pathname. If you omit *vob-selector*, the default is the current working directory

unless the reference page specifies otherwise. Specify *vob-selector* in the form [**vob:**]*pname-in-vob* (for some commands, the **vob:** prefix is required)

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

Object Names

In object-creation commands, you must compose the object name according to these rules:

- It must contain only letters, digits, and the special characters underscore (_), period (.), and hyphen (-). A hyphen cannot be used as the first character of a name.
- It must not be a valid integer or real number. (Be careful with names that begin with "0x", "0X", or "0", the standard prefixes for hexadecimal and octal integers.)
- It must not be one of the special names " **.** ", " **..** ", or " **...** ".

EVENT RECORDS AND COMMENTS

Each change to a VOB is recorded in an event record in the VOB database. Many **multitool** commands include options you can use to include a comment string in the event record created by the command. Commands that display event record information (**describe**, **lsepoch**, **lspacket**, **lsreplica**, **lstype**) show the comments. See the **fmt_ccase** reference page in the *Command Reference* for a description of the report-writing facility built in to these commands.

All commands that accept comment strings recognize the same options:

–c·omment *comment*
Specifies a comment for all event records created by the command. The comment string must be a single command-line token; typically, you must enclose it in double quotes.

–cfi·le *comment-file-pname*
Specifies a text file whose contents are to be placed in all event records created by the command.

NOTE: A final line terminator in this file is included in the comment.

–cq·uery
Prompts for one comment, to be placed in all the event records created by the command.

–cqe·ach
For each object processed by the command, prompts for a comment to be placed in the corresponding event record.

–nc·omment
(No additional comment) For each object processed by this command, creates an event record with no comment string.

A **–cq** or **–cqe** comment string can span several lines. To end a comment, enter an **EOF** character at the beginning of a line, by typing a period character (**.**) and pressing ENTER, by typing CTRL+D on UNIX, or by typing CTRL+Z ENTER on Windows. For example:

cleartool checkout main.c
```
Checkout comments for "main.c":
```
This is my comment; the following line terminates the comment.
.
```
Checked out "main.c" from version "\main\3"
```

The **cleartool chevent** command revises the comment string in an existing event record. See the **events_ccase** reference page in the *Command Reference* for more information about event records.

Specifying Comments Interactively

multitool can reuse a comment specified previously. If the environment variable **CLEARCASE_CMNT_PN** specifies a file, that file is used as a comment cache:

- When a **multitool** subcommand prompts for a comment, it offers the current contents of file **$CLEARCASE_CMNT_PN** (UNIX) or **%CLEARCASE_CMNT_PN%** (Windows) as the default comment.
- When you specify a comment string interactively, the **multitool** subcommand updates the contents of **CLEARCASE_CMNT_PN** with the new comment. (The comment cache file is created if necessary.)

  NOTE: A comment that is specified noninteractively (for example, with the command **mkreplica –export –c "new replica for buenosaires"**) does not update the comment cache file.

The value of **CLEARCASE_CMNT_PN** can be any valid pathname. Using a simple file name (for example, **.msite_cmnt**) implements a comment cache for the current working directory; different directories can have different **.msite_cmnt** files. Using the full pathname **%HOME%\.msite_cmnt** (on Windows) or **$HOME/.msite_cmnt** (on UNIX) implements a cache of the individual user's comments across all ClearCase VOBs.

Customizing Comment Handling

Each command that accepts a comment string has a comment default, which takes effect if you enter the command without any comment option. For example, the **restorereplica** command's comment default is **–cqe**, so you are prompted to enter a comment for each replica being restored. The **rmreplica** command's comment default is **–nc**: remove the replica without prompting for a comment.

You can define a default comment option for each **multitool** command with a user profile file, **.clearcase_profile**, in your home directory. For example, you can establish **–cqe** as the comment default for the **chmaster** command. See the **comments** and **profile_ccase** reference pages in the *Command Reference* for details.

recoverpacket

Resets epoch row table so changes in lost packets are resent

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

recoverpacket [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**] [**–sin·ce** *date-time*] *replica-selector* ...

DESCRIPTION

The **recoverpacket** command resets the epoch row at a sending replica to reflect the last synchronization sent to a receiving replica before a particular time. It scans through a list of epoch rows saved at the time of each export, looking for an entry prior to the time specified. When it finds an entry, it uses the associated row to reset the epoch row for the specified receiving replica. The next time a packet is sent, it includes the changes that were in the lost packet.

Resetting Epoch Numbers Automatically

When you send an update packet to another replica, success of the transport and import phases is assumed. Therefore, the sending replica's epoch number matrix is updated to reflect that the changes are made at the receiving replica. However, if the packet is lost before reaching the receiving replica, the sending replica's assumption that the receiving replica is up to date is incorrect.

The updated epoch numbers must be returned to the values they had before the packet was sent. Making these corrections to the sending replica's epoch number matrix causes it to include the same changes in the next update packet it sends to the receiving replica.

The administrator at the receiving replica must run an **lshistory** command to determine the time of the last successful import. The administrator at the sending replica uses this time in the **recoverpacket** command.

NOTE: If the two sites are not in the same time zone, or you do not send packets at the same time you generate them (for example, you generate packets at midnight and send them at 6:00 A.M.), you must adjust for the time difference.

Resetting Epoch Numbers Manually

If there are no saved epoch rows for the replica that are as old as the specified time, the **recoverpacket** command fails. In this case, the administrator at the receiving site must use the **lsepoch** command to determine the correct epoch number, and the administrator at the sending site must run **chepoch** on the sending replica to reset the epoch row. See the **chepoch** reference page for more information.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB.

Mastership: No mastership restrictions.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SPECIFYING THE TIME. *Default:* If the time is not specified, **recoverpacket** uses the current time (and, therefore, resets the epoch row so that the changes in the most recent update packet are resent).

–sin·ce *date-time*
Specifies the time of the last successful processing of a packet at the receiving site. The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**
where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** | **yesterday** | **Sunday** | ... | **Saturday** | **Sun** | ... | **Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** | ... | **December** | **Jan** | ... | **Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1990
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

SPECIFYING THE ROW TO BE MODIFIED. *Default:* None.

replica-selector

Specifies the replica for which the epoch row is reset. Specify *replica-selector* in the form [**replica:**]*replica-name*[*@vob-selector*]

replica-name Name of the replica (displayed with **lsreplica**)

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

- Reset the epoch row for replica **sanfran_hub** so that changes sent since last Monday will be included in the next packet that is sent.

  multitool recoverpacket –nc –since Monday sanfran_hub

- Reset the epoch row for replica **boston_hub** so that the changes included in the most recent update packet will be included in the next packet that is sent.

  multitool recoverpacket –c "send latest packet" boston_hub@\dev

- Determine the last successful import at replica **bangalore**, reset the epoch row at replica **boston_hub**, and send an update packet.

  At replica **bangalore**:

  cleartool lshistory replica:bangalore@\dev
  ```
  19-Oct.15:36 smg import sync from replica "boston_hub" to replica
  "bangalore"
  "Imported synchronization information from replica "boston_hub".
  ...
  ```

  At replica **boston_hub** (remember to adjust for the time zone difference):

  multitool recoverpacket –since 19-Oct.05:06 bangalore@/vobs/dev
  ```
  Using epoch information from Monday 10/19/99 05:05:45
  Epoch row for replica "bangalore" successfully reset.
  ```

  multitool syncreplica –export –fship bangalore@/vobs/dev
  ```
  Generating synchronization packet
  /usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_22-Oct-99.15.44.28_48
  96_1
  - shipping order file is
  /usr/atria/shipping/ms_ship/outgoing/sh_o_sync_boston_hub_22-Oct-99.15.44.
  28_4896_1
  Attempting to forward/deliver generated packets...
  -- Forwarded/delivered packet
  /usr/atria/shipping/ms_ship/outgoing/sync_boston_hub_22-Oct-99.15.44.28_48
  96_1
  ```

SEE ALSO

chepoch, **lsepoch**, **restorereplica**
VOB Operations and the Oplog in Chapter 1, *Introduction to MultiSite*

reqmaster

Sets access controls for mastership requests or requests mastership of a branch or branch type

APPLICABILITY

Product	Command type
ClearCase	cleartool subcommand
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

- Display or set the ACL for mastership requests:

  reqmaster –acl [**–edi·t** | **–set** *pname* | **–get**] *vob-selector*

- Set access controls for the replica, branches, or branch types:

  reqmaster [**–c·omment** *comment* | **–cq·uery** | **–nc·omment**]
  { { **–enable** | **–dis·able** } *vob-selector*
  | { **–den·y** | **–allow** } [**–inst·ances**] *branch-type-selector ...*
  | { **–den·y** | **–allow** } *branch-pname ...*
  }

- Request mastership of a branch or branch type:

  reqmaster [**–c·omment** *comment* | **–cq·uery** | **–nc·omment**]
  [**–lis·t**] { [*branch-pname ...*] [*branch-type-selector ...*] }

DESCRIPTION

This command has three forms: two forms to configure access controls for mastership requests and one form to request mastership of a branch or branch type from the replica that masters the object. For more information, see Chapter 9, *Implementing Requests for Mastership* in the *Administrator's Guide* for Rational ClearCase MultiSite.

SETTING ACCESS CONTROLS

To allow requests for mastership, the MultiSite administrator must set access controls at each replica:

- Add developers to the replica's access control list (ACL). Use the **–acl** option with **–edit** or **–set** to edit the ACL.
- Enable replica-level access. By default, replica-level access is not enabled. To enable it, use the **–enable** option.

Also, the type and the object must allow mastership requests. By default, type-level and object-level access are enabled. You can enable replica-level access, but deny requests for mastership of specific branches, specific branch types, or all branches of a specific type. Even if replica-level access is enabled, the **reqmaster** command fails if requests for mastership are denied at the type level or object level. Use the **–deny** option to deny requests at the type and object level.

REQUESTING MASTERSHIP OF A BRANCH OR BRANCH TYPE

This form of the **reqmaster** command contacts a sibling replica and requests that the replica transfer mastership to the current replica. You can also use **reqmaster** to display information about whether a mastership request will succeed.

If you specify multiple branches or branch types and the request fails for one or more items, **reqmaster** prints error messages for the failures and continues processing the other items.

TROUBLESHOOTING

If the **reqmaster** command fails, the error message indicates whether the failure occurred at the current replica or the sibling replica.

If the **reqmaster** command fails with the message `can't get handle`, enter the command again. If it continues to fail, ask the administrator of the sibling replica to check the status of the VOB server.

When you request mastership, the **reqmaster** command may complete successfully, but the mastership is not transferred to your current replica. In this case, verify that the synchronization packet was sent from the sibling replica and that your current replica imported it successfully.

Errors that occur during the mastership request process, including errors occurring during the synchronization export, are written to the **msadm** log file. To view this log, use the **cleartool getlog** command or the ClearCase Administration Console (Windows).

For more information on error messages from the **reqmaster** command, see Chapter 9, *Implementing Requests for Mastership* in the *Administrator's Guide* for Rational ClearCase MultiSite.

RESTRICTIONS

Setting Access Controls

Identities: To set the ACL, you must have write permission on the ACL or have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

To enable mastership requests at the replica level, you must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: No locks apply.

Mastership: The replica must be self-mastering. For you to allow or deny mastership requests for a branch or branch type, your current replica must master the object.

Requesting Mastership of a Branch:

Identities: You must be on the replica's ACL.

Locks: An error occurs if one or more of these objects are locked: branch, branch type, VOB.

Mastership: Your current replica must not master the branch.

Other: An error occurs in any of the following cases:

- Mastership requests are denied at any of the following levels: replica, type object, object.
- There are checkouts on the branch (except for unreserved, nonmastered checkouts).
- You specify a branch associated with a stream.

Requesting Mastership of a Branch Type:

Identities: You must be on the replica's ACL.

Locks: An error occurs if one or more of these objects are locked: branch type, VOB, branch instances that have default mastership.

Mastership: Your current replica must not master the branch type.

Other: An error occurs in any of the following cases:

- Mastership requests are denied at any of the following levels: replica, type object, any branch type instances with default mastership.
- There are checkouts on any branch type instances with default mastership (except for unreserved, nonmastered checkouts).
- You specify a branch type associated with a stream.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *Customizing Comment Handling* in the **multitool** reference page. To edit a comment, use **chevent**.

–c·omment *comment* | **–cq·uery** | **–nc·omment**
Overrides the default with the specified comment option.

DISPLAYING OR SETTING ACCESS CONTROLS. *Default:* None. You must specify access controls. Specifying **–acl** with no other option displays the ACL for the current replica in the VOB family specified by *vob-selector*.

–acl [**–edi·t** | **–set** *pname* | **–get**] *vob-selector*
By default or with **–get**, displays the ACL for the current replica in the VOB family specified by *vob-selector*. With **–edit**, opens the ACL for the current replica in the editor specified by (in order) the **WINEDITOR** (UNIX), **VISUAL**, or **EDITOR** environment variable. With **–set**, uses the contents of *pname* to set the ACL for the current replica.

Specify *vob-selector* in the form **vob:***pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–enable *vob-selector*
Allows mastership requests to be made to the current replica in the VOB family specified by *vob-selector*.

–**dis·able** *vob-selector*
Denies all mastership requests made to the current replica in the VOB family specified by *vob-selector*.

{ **–deny** | **–allow** } [**–inst·ances**] *branch-type-selector* ...
Denies or allows requests for mastership of the specified branch type. With **–instances**, denies or allows requests for mastership of all branches of the specified type. Specify *branch-type-selector* in the form **brtype:***type-name*[@*vob-selector*]

type-name	Name of the branch type
vob-selector	VOB specifier; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob	Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

{ **–deny** | **–allow** } *branch-pname* ...

Denies or allows requests for mastership of the specified branch object. Specify *branch-pname* in the form *file-pname@@branch*. For example:

```
cmdsyn.c@@/main/v3.8
header.h@@\main\v1\bugfix
```

REQUESTING MASTERSHIP. *Default:* Sends a request for mastership to the master replica of the object.

–lis·t

Does not request the mastership change; instead, displays information about whether a request would succeed.

branch-pname

Branch whose mastership you are requesting. For example:

```
cmdsyn.c@@/main/v3.8
header.h@@\main\v1\bugfix
```

branch-type-selector

Branch type whose mastership you are requesting. For example:

```
brtype:v2.0_integration@vob:\tests
```

EXAMPLES

- Display the ACL for the current replica in the VOB family **/vobs/dev**, and then change it to give full access to **ccadmin** and permission to request mastership to **gail** and **paul**.

  multitool reqmaster –acl –get vob:/vobs/dev

  ```
  # Replica boston_hub@/vobs/dev
  # Request for Mastership ACL:
  Everyone: Read
  ```

  cat > /tmp/boston_hub_aclfile

  ```
  # Replica boston_hub@/vobs/dev
  # Request for Mastership ACL:
  User:purpledoc.com/ccadmin Full
  User:purpledoc/ccadmin Full
  User:purpledoc.com/gail Change
  User:purpledoc/gail Change
  User:purpledoc.com/paul Change
  User:purpledoc/paul Change
  ```

  multitool reqmaster –acl –set /tmp/boston_hub_aclfile vob:/vobs/dev

multitool reqmaster –acl –get vob:/vobs/dev

```
# Replica boston_hub@/vobs/dev
# Request for Mastership ACL:
User:purpledoc.com/ccadmin Full
User:purpledoc/ccadmin Full
User:purpledoc.com/gail Change
User:purpledoc/gail Change
User:purpledoc.com/paul Change
User:purpledoc/paul Change
```

- Allow requests for mastership for all branches and branch types mastered by the current replica in VOB family **\tests**, except for the branch type **v2.0_integration** and all branches of that type.

  multitool reqmaster –enable vob:\tests

  ```
  Requests for mastership enabled in the replica object for "vob:\tests"
  ```

  multitool reqmaster –deny –instances brtype:v2.0_integration@vob:\tests

  ```
  Requests for mastership denied for all instances of
  "brtype:v2.0_integration@vob:\tests"
  ```

  multitool reqmaster –deny brtype:v2.0_integration@vob:\tests

  ```
  Requests for mastership denied for branch type
  "brtype:v2.0_integration@vob:\tests"
  ```

- Allow requests for mastership for all branches and branch types mastered by the current replica in VOB family **\dev**, except for the branch **cmdsyn.m@@\main\v1.0_bugfix**.

  multitool reqmaster –enable vob:\dev

  ```
  Requests for mastership enabled in the replica object for "vob:\dev"
  ```

  multitool reqmaster –deny \dev\cmdsyn.m@@\main\v1.0_bugfix

  ```
  Requests for mastership denied for branch
  "\dev\cmdsyn.m@@\main\v1.0_bugfix"
  ```

- Deny requests for mastership for all branches and branch types mastered by the current replica.

  multitool reqmaster –disable vob:/vobs/dev

  ```
  Requests for mastership disabled in the replica object for "vob:/vobs/dev"
  ```

- Deny requests for mastership of the branch type **v2.0_integration**.

  multitool reqmaster –deny brtype:v2.0_integration@vob:\tests

  ```
  Requests for mastership denied for branch type
  "brtype:v2.0_integration@vob:\tests"
  ```

- Display mastership information about the branches **include.h@@\main\integ** and **acc.c@@\main**.

  multitool reqmaster –list include.h@@\main\integ acc.c@@\main

- Request mastership of the branch **cmdsyn.m@@/main/v2.6_dev**.

  multitool reqmaster cmdsyn.m@@/main/v2.6_dev
- Request mastership of the branch type **v2.0_integration**.

  multitool reqmaster brtype:v2.0_integration@vob:\tests

SEE ALSO

chmaster

restorereplica

Restores VOB replica from backup

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

restorereplica [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**] [**–f·orce**] [**–override**] [**–invob** *vob-selector* | [**–rep·lace**] *replica-selector* ...]

DESCRIPTION

Execute this command IMMEDIATELY after restoring a VOB replica from backup. Proceeding with normal development (and generating new changes) at a restored replica before executing this command can lead to IRREPARABLE inconsistencies among the replicas in a VOB family.

restorereplica replaces missing changes in a VOB replica that has been restored from backup, as follows:

1. Causes the current replica to create special update packets that contain update requests to other replicas.
2. Locks the current replica's VOB object and marks the replica as being in the process of restoration.
3. Increments the recovery incarnation for the replica.
4. Causes **lsreplica –long** to indicate which replicas must send restoration updates to the current replica.

The current replica remains in the restoration state until your site has received and applied (using **syncreplica –import**) all the restoration updates needed to bring the replica up to date with the state of the VOB family. Collectively, these updates include all the changes to the VOB family since the backup was made, including changes made in the current replica before its failure.

During the process of restoration, the **lsreplica –long** command annotates its listing to indicate which replicas must send restoration updates to the replica.

For a description of the replica restoration procedure, see *Restoring and Replacing Replicas* on page 190.

LOCKING OF THE REPLICA

restorereplica locks the current replica's VOB object. This ensures that while restoration proceeds through execution of **syncreplica –import** commands, no other changes are made to the current replica.

When **syncreplica** applies the final required update, it displays a message indicating that the restoration process is complete. At this point, use the **cleartool unlock vob:** command to unlock the restored VOB replica, enabling normal development to proceed.

OPTIMIZING THE RESTORATION PROCESS

By default, **restorereplica** requires that the replica receive restoration updates from all other replicas in its VOB family (either directly or indirectly). Only after all the updates are imported does the **syncreplica** command display the message indicating that restoration is complete.

In some cases, you can relax this requirement without compromising the correctness of the restoration process. The replica will be brought up to date if it receives a restoration update from only one replica—the last one to which the replica sent an update before it was restored from the backup version. You can specify the name of that last-updated replica (or a list of replicas, one of which must be the last-updated one) to **restorereplica**. **syncreplica** displays the restoration-completed message after receiving restoration updates from all the specified replicas.

CAUTION: Making a mistake in using this optimization can make the restored replica irreparably inconsistent with other replicas.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: No locks apply.

Mastership: No mastership restrictions.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–cqe**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment-string* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SUPPRESSING INTERACTIVE PROMPTS. *Default:* **restorereplica** prompts you for confirmation.

–f·orce
Suppresses the confirmation step.

SPECIFYING THE VOB FAMILY. *Default:* Processes the replica that contains the current working directory.

–invob *vob-selector*
Processes the current replica in the specified VOB family. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

REDUCING THE NUMBER OF REQUIRED UPDATES. *Default:* The replica requires restoration updates from *all* other members of its VOB family. The **syncreplica** command declares the VOB to be restored completely only after all the updates have been processed.

CAUTION: Incorrect use of these options allows new changes to be made to the replica before all missing changes are received from other replicas. This may place the entire VOB family in an irreparably inconsistent state.

replica-selector ...
Specifies a subset of replicas from which updates are required before **syncreplica** declares the VOB to be restored completely. Specify *replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*]

replica-name Name of the replica (displayed with **lsreplica**)

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

–rep·lace *replica-selector* ...

Changes the subset of replicas from which restoration updates are required.

–override

Overrides normal restoration processing and declares the VOB to be restored completely. The **lsreplica –long** command no longer annotates any replicas as needing to provide updates, and you can use **cleartool unlock vob:** to place the replica back in normal service.

When you specify this option, the command displays a list of replicas from which updates have not been received and prompts you to cancel the operation or continue.

EXAMPLES

For an example of restoring a replica, see *Restoring and Replacing Replicas* on page 190.

SEE ALSO

chepoch, **lsepoch**, **lsreplica**, **syncreplica**

rmreplica

Deletes a VOB-replica object

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

rmreplica [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**] *replica-selector*

DESCRIPTION

CAUTION: To delete a replica, you must complete all steps described in *Deleting a Replica* on page 119. If you do not complete all steps in the correct order, synchronization and mastership problems can occur in other replicas in the VOB family.

This command deletes from the current replica's database the VOB-replica object that records the existence and identity of another replica. Typically, you use this command at your site to record the fact that a replica at another site has been decommissioned and deleted.

NOTE: If executing this command makes the current replica the last remaining member of the VOB family, **rmreplica** turns off operation logging for this VOB and removes all operation logs, which may take a long time.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB, replica.

Mastership: Your current replica must master the replica being removed.

Other: The following restrictions apply:

- You cannot delete your current replica's VOB-replica object.
- You cannot delete a replica if your current replica considers it to master one or more objects.

OPTIONS AND ARGUMENTS

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SPECIFYING THE REPLICA. *Default:* None.

replica-selector
Specifies the VOB-replica object to be deleted from the current replica's database. Specify *replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*]

replica-name Name of the replica (displayed with **lsreplica**)

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EXAMPLES

- Remove the VOB-replica object that records the existence of replica **tokyo** from the database of the current replica.

  multitool rmreplica tokyo
  ```
  Deleted replica "tokyo".
  ```

SEE ALSO

chmaster, **mkreplica**

shipping.conf

Store-and-forward configuration file

APPLICABILITY

Product	Command type
MultiSite	MultiSite data structure

Platform
UNIX

SYNOPSIS

/var/adm/atria/config/shipping.conf

DESCRIPTION

This file controls the operation of the MultiSite store-and-forward facility on each host. The file consists of comment lines (starting with **#**) and one or more configuration entries.

The **shipping.conf** file can contain the configuration entries described below. In some cases, the corresponding store-and-forward operation fails if an entry is missing; in other cases, there is a hard-coded default.

MultiSite installation creates the file *ccase-home-dir*/**config/services/shipping.conf.template**, in which all these entries are defined. If **/var/adm/atria/config/shipping.conf** does not exist, the installation creates it by copying the template file. If **/var/adm/atria/config/shipping.conf** exists, the installation advises you to compare the existing file to the template and make any necessary changes.

NOTE: If you do not install ClearCase and MultiSite in the default installation directory (**/usr/atria**), you must edit the **shipping.conf** file and change **/usr/atria** to the pathname of your installation directory.

PACKET SIZE

MAX-DATA-SIZE *size* [**k** | **m** | **g**]

Controls the splitting of individual logical packets into multiple physical packets. Limiting the size of physical packets can improve the reliability of packet delivery in some networks. The *size* integer (with the optional **k**, **m**, or **g** suffix) specifies the maximum size for a physical packet file. **k** specifies KB (kilobytes); **m** specifies MB (megabytes); **g** specifies GB (gigabytes). Omitting the suffix specifies KB. To specify no limit, use 0 (zero).

This value is used by the following commands (unless you also specify **–maxsize**):

- **mkreplica –fship**
- **mkreplica –ship**
- **syncreplica –fship**
- **syncreplica –ship**
- **sync_export_list**

When you invoke **mkreplica** or **syncreplica** with **–out** or **–tape**, this value is not used and you must use **–maxsize** to limit the packet size.

Default: 2097151k

NOTIFICATION

NOTIFICATION-PROGRAM *e-mail-program-pathname*

The electronic mail program to be invoked in these circumstances:

- When **shipping_server** finds that a shipping order it is about to process has expired
- When an undeliverable packet is returned to the original sending host by another host's **shipping_server** (see the description of **EXPIRATION**)
- When **syncreplica –import** finds a replica-creation packet, which must be processed with a **mkreplica** command

The mail program is invoked as follows:

e-mail-program-pathname **–s** *subject* **–f** *message-file addr* . . .

Default: **/usr/atria/bin/notify**. This program is also used if no **NOTIFICATION-PROGRAM** entry exists.

ADMINISTRATOR ADDRESS

ADMINISTRATOR *e-mail-address*

The electronic mail address of the administrator who administers the store-and-forward facility on the local host.

A mail message is sent to the specified address in the circumstances listed in *NOTIFICATION*. The configuration file can contain multiple **ADMINISTRATOR** entries; messages are sent to all the specified mail addresses.

Default: **root**

STORAGE BAY AND RETURN BAY

STORAGE-BAY *storage-class directory-pathname*
RETURN-BAY *storage-class directory-pathname*

These lines define storage bay and return bay directories. A storage bay holds the outgoing and incoming update packets and shipping orders of a particular storage class. A return bay holds incoming or outgoing packets in the process of being returned to their origin because they could not be delivered to all specified destinations.

You can use multiple **STORAGE-BAY** and **RETURN-BAY** entries to define multiple bays for a particular storage class. **shipping_server** selects one of the bays for each packet based on the available disk space in the bays' disk partitions. The order in which you specify storage bays does not matter.

MultiSite installation establishes a default storage bay and return bay on the local host in the **/var/adm/atria/shipping** directory. Each bay contains subdirectories named **incoming** and **outgoing**, which hold incoming and outgoing packets. Shipping operations look for packets in these subdirectories. Before using the store-and-forward facility, make sure that the disk partition where the shipping directory is created has sufficient free space for anticipated replica-creation and update packets.

You must create *directory-pathname* with a standard UNIX **mkdir** command. You must also create the **incoming** and **outgoing** directories within the new bay. Packets placed in a bay are assigned the same owner, groups, and read-write permissions as the bay itself. (Execute permission and any special permissions on the bay are ignored.) Be sure to adjust these permissions (if necessary) to enable successful execution of MultiSite commands to process the packets, and to guard against unauthorized access.

NOTE: The **incoming** and **outgoing** directories must be on the same file system.

Default: The **–default** storage class is used for packets that are not assigned to any storage class, and for packets whose storage class is not configured.

```
STORAGE-BAY -default /usr/atria/shipping/ms_ship

RETURN-BAY -default /usr/atria/shipping/ms_rtn
```

EXPIRATION PERIOD

EXPIRATION *storage-class number-of-days*
EXPIRATION –default *number-of-days*

Specifies the expiration period (in days) for shipping orders generated in the specified storage class. This period begins at the time the shipping order is generated. If a packet cannot be delivered to all of its specified destinations, the packet is returned to the original sending host and one or more electronic mail messages are sent (see the descriptions in the sections *ADMINISTRATOR ADDRESS* and *NOTIFICATION*).

Specifying **–default** as the storage class sets the expiration period for shipping orders that are not assigned to any storage class, and for shipping orders whose storage class is not configured.

A zero **EXPIRATION** value specifies no expiration and delivery is reattempted indefinitely.

This setting is overridden by the **–pexpire** option to **syncreplica** or **mkreplica**.

The **shipping_server** program does not retry delivery of a packet. The **EXPIRATION** specification is useful only if you schedule periodic invocations of **sync_export_list –poll** to attempt delivery of any undelivered packets.

Default: 14 days.

PACKET ROUTING

ROUTE *next-hop host* ...
ROUTE *next-hop* **–default**

Controls the network routing of packets. Packets whose final destination is any of the *host* arguments are sent to the host named *next-hop*. This host is responsible for final delivery of the packet to its destinations (or additional forwarding). *next-hop* and *host* can be host names (which must be usable by hosts in different domains) or numeric IP addresses.

You can include multiple **ROUTE** entries in the configuration file. The special keyword **–default** accommodates all hosts that are not specified in another **ROUTE** entry.

Default: None.

RECEIPT HANDLER

RECEIPT-HANDLER *storage-class script-pathname*

Specifies a script for the **shipping_server** to run for each packet received in a shipping bay. By default, no script is specified. We recommend that you specify the **sync_receive** script as the **RECEIPT-HANDLER** entry. For example:

```
RECEIPT-HANDLER  -default /usr/atria/config/scheduler/tasks/sync_receive
```

For each packet that is received, **shipping_server** handles it as follows:

1. Reads the **shipping.conf** file to find the appropriate **RECEIPT-HANDLER** entry for the packet.
   - If the packet is associated with a storage class and there is a **RECEIPT-HANDLER** entry for that storage class, **shipping_server** uses the *script-pathname* specified in that entry.
   - If the packet is not associated with a storage class and there is a **RECEIPT-HANDLER** value for the **–default** storage class, **shipping_server** uses the script specified for **–default**.
2. Invokes the receipt handler as follows:

   script-pname [**–d·ata** *packet-file-pname*] [**–a·ctual** *shipping-order-pname*]
   [**–s·class** *storage-class*] **–o·rigin** *hostname*

   where

script-pname	Script specified in the **RECEIPT-HANDLER** entry.
–d·ata *packet-file-pname*	Location of the packet. This parameter is used only when the packet is destined for this host.
–a·ctual *shipping-order-pname*	Location of the shipping order. This parameter is used only when the packet is destined for another host.
–s·class *storage-class*	Storage class associated with the packet. This parameter is used only if the packet was associated with a storage class when it was created.
–o·rigin *hostname*	Host name of the machine from which the packet was first sent.

NOTE: If a packet is destined for both the local host and another host, both the **–data** and **–actual** parameters are used. The packet is imported at the replica on the host, and then forwarded to its next destination.

Default: None.

PORT NUMBERS

CLEARCASE_MIN_PORT *port-number*
CLEARCASE_MAX_PORT *port-number*

CAUTION: Set these entries only on hosts that can communicate through the firewall and have been installed with the MultiSite **shipping_server**-only option.

These entries specify the range of ports for **shipping_server** to use on a firewall system, and they are set as environment variables in the **shipping_server** environment.

Guidelines for setting the values:

- The value range for **CLEARCASE_MIN_PORT** is 1024 through 65534.
- The value range for **CLEARCASE_MAX_PORT** is 1025 through 65535.
- The value of **CLEARCASE_MAX_PORT** must be greater than the value of **CLEARCASE_MIN_PORT**.
- We recommend that you use the range 49152 through 65535, which is the Dynamic/Private Port Range. If you use a value within the Registered Ports range (1024 through 49151), the **shipping.conf** parser prints an informational message.

NOTE: To use **shipping_server** on a firewall system, you must also set the **CLEARCASE_MIN_PORT** and **CLEARCASE_MAX_PORT** environment variables in the **atria_start** script. For more information, see *Specifying Port Values* on page 101.

Default: None.

shipping_server

Store-and-forward packet transport server

APPLICABILITY

Product	Command type
MultiSite	MultiSite command

Platform
UNIX
Windows

SYNOPSIS

shipping_server [**–scl·ass** *storage-class-name*] { **–pol·l** | *sources* ... }

This command is located in *ccase-home-dir*/**etc** on UNIX and *ccase-home-dir***bin** on Windows.

DESCRIPTION

This command processes one or more shipping orders on the local host, causing the associated packets or files to be sent to remote sites.

After delivering the data file specified in a shipping order to all its destinations, **shipping_server** deletes the data file unless one of the destinations is the local host.

NOTE: When **shipping_server** starts processing a shipping order, it locks the order. This prevents subsequent invocations of **shipping_server** from processing the order.

TCP/IP Connection

To transmit a file, **shipping_server** uses UDP to contact the **albd_server** process on the receiving host, and **albd_server** invokes **shipping_server** in receive mode on the receiving host.

If you are sending packets through a firewall (that is, the CLEARCASE_MIN_PORT and CLEARCASE_MAX_PORT environment variables are set), **shipping_server** tries to use TCP to contact the remote **albd_server**. If that connection fails, **shipping_server** uses UDP. For more information, see *Installing Store-and-Forward on a UNIX Firewall Host* on page 98.

On UNIX, **shipping_server** forks one subprocess for each packet that it needs to send. As many as 10 separate **shipping_server** subprocesses, each trying to send a single packet, can be started for each invocation of **shipping_server**. The same number of subprocesses are forked on the

receiving machine. As a subprocess finishes, another can be started, but only 10 can be active simultaneously.

After a TCP connection is established between the two **shipping_server** processes, they transfer the file. The receiving **shipping_server** selects a storage bay using the local store-and-forward configuration settings. See **shipping.conf** (UNIX) or the **MultiSite Control Panel** (Windows). If a storage class is assigned multiple storage bays, available disk space determines the selection of a bay.

UNIX: The packet file is created with the same owner and group as the storage bay directory, and its access mode is taken from the directory's read and write permissions. (The execute permission and special permissions, if any, are ignored.)

Windows: The packet file inherits permissions from the Windows ACL on the storage bay directory.

Colon Characters in Packet Names

If a packet name contains a colon (**:**), **shipping_server** changes the colon to a period (**.**) during processing. This change allows packets to be delivered to Windows machines, which do not allow colons within file names.

Handling of File Name Conflicts

You can use the **mkorder** and **shipping_server** commands to transmit arbitrary files if the files are located in the same directory as their associated shipping orders. If a file with the same name already exists on the receiving host, the new file is renamed to *filename*_**1** (if you send another file with the same name, it is renamed to *filename*_**2**, and so on).

Log File

UNIX: shipping_server writes records of all packets sent and received, along with all errors, to file **/var/adm/atria/log/shipping_server_log**.

Windows: shipping_server writes records of all packets sent and received, notification messages, and all errors to the Windows event viewer. It writes log messages to file *ccase-home-dir***var\log\shipping_server_log**.

RESTRICTIONS

Identities: You must have write and execute permissions on the directory containing the shipping order. On UNIX, you must own the data file or be **root**.

Locks: No locks apply.

Mastership: No mastership restrictions.

Other: The shipping order and the data file it specifies must be located in the same directory.

OPTIONS AND ARGUMENTS

RESTRICTING PROCESSING TO A STORAGE CLASS. *Default:* Processes all shipping orders specified or found in a search.

–scl·ass *storage-class-name*
Processes shipping orders for the specified storage class only.

SPECIFYING THE SHIPPING ORDERS. *Default:* None.

–pol·l
Processes shipping orders located in some (if you use **–sclass**) or all MultiSite storage bays defined in the **shipping.conf** configuration file on UNIX or the **MultiSite Control Panel** on Windows.

NOTE: **shipping_server** processes only shipping orders whose file names start with the characters "`sh_o_`". If you create shipping orders, name them according to this convention, or omit the **–poll** option and specify the shipping order pathnames.

On UNIX, only shipping order files that you own are processed. (EXCEPTION: when **root** runs this program, shipping order files are processed regardless of ownership.)

sources ...
One or more pathnames of files and/or directories. Each file you specify is processed if it contains a valid shipping order. For each directory you specify, **shipping_server** processes some (if you use **–sclass**) or all shipping orders stored in that directory.

EXAMPLES

- Process all shipping orders in all MultiSite storage bays.

  shipping_server –poll

- Process a particular shipping order. Note that the pathname argument specifies the shipping order file, not the data file to be transmitted.

  **/usr/atria/etc/shipping_server **
  /var/adm/atria/shipping/ms_ship/sh_o_sync_sydney_19-May-99.09:48:45_7660_1

- Process all shipping order files in a specified directory.

  shipping_server "c:\Program Files\Rational\ClearCase\var\shipping\ms_ship"

- Process all shipping orders in the storage bays of a specified storage class.

  /usr/atria/etc/shipping_server –poll –sclass daily

SEE ALSO

mkorder, **MultiSite Control Panel**, **shipping.conf**, **syncreplica**, **sync_export_list**
Chapter 10, *Troubleshooting MultiSite Operations*

sync_export_list

Generates and sends update packets

APPLICABILITY

Product	Command type
MultiSite	MultiSite command

Platform
UNIX
Windows

SYNOPSIS

- Generate update packets:

  sync_export_list [**–c·ompress**] [**–lo·gdir** *log-directory*]
  [**–f·ship** | **–sh·ip**] [**–lockwait** *minutes*] [**–q·uiet** *mode*]
  [**–wo·rkdir** *directory*] [**–m·axsize** *max-packet-size*]
  [**–sc·lass** *storage-class*] [**–u·pdate**] [**–li·mit** *num-packets*]
  [**–t·race**] [**–p·oll**] [**–i·terate** *num-tries* [**–wa·it** *num-seconds*]]
  { **–a·ll** | **–r·eplicas** *replica-list* [*script-file*] | *script-file* }

- Process shipping orders in the host's storage bays:

  sync_export_list –p·oll [**–sc·lass** *storage-class*]

- Print help on command options:

  sync_export_list –h·elp

On UNIX, **sync_export_list** is located in *ccase-home-dir*/**config/scheduler/tasks**. On Windows, **sync_export_list** is located in *ccase-home-dir***config\scheduler\tasks**.

DESCRIPTION

sync_export_list generates update packets for one or more VOB replicas. You can specify options for packet generation and transport on the command line, in a script file, or by using a combination of the command line and a script file.

You can run **sync_export_list** manually, or run it automatically with the **schedule** command. For more information, see the **schedule** reference page in the *Command Reference*.

RETRYING SYNCHRONIZATION WHEN THE VOB IS LOCKED

By default, synchronization exports fail if the VOB is locked. To allow **sync_export_list** to retry an export when it encounters a lock, use the **–lockwait** option, which specifies the amount of time (in minutes) for **sync_export_list** to keep trying to write to the VOB. During that time, **sync_export_list** retries the write operation every minute. If the time elapses and the VOB is still locked, **sync_export_list** exits with an error.

The **–lockwait** option sets the **CLEARCASE_VOBLOCKWAIT** environment variable in the script's environment. If **–lockwait** is not used, **sync_export_list** ignores **CLEARCASE_VOBLOCKWAIT** if it is set outside the script's environment.

NOTE: **sync_export_list** waits only if it detects the lock before it starts processing oplogs. If an administrator locks the VOB during oplog processing, **sync_export_list** exits with an error.

CONFIGURATION FILE

You can modify the behavior of the **sync_export_list** script by creating a file named **MSimport_export.conf** and setting values in it. On UNIX, create the file in the directory **/var/adm/atria/config**. On Windows, create the file in the directory *ccase-home-dir***var****config**.

The file can include the following export setting:

disable_export_locking = 1

Disables use of the export lockfile, allowing multiple exports from a single replica to run simultaneously. Setting the value to **0** (default) enables use of the lockfile.

This setting and the **–lockwait** option are not related. This setting configures use of the lock created by the **sync_export_list** process to prevent interference among export processes, and the **–lockwait** option handles ClearCase VOB locks.

TROUBLESHOOTING

sync_export_list fails if there is another **sync_export_list** process exporting data from the same replica, unless export locking is disabled (see *CONFIGURATION FILE*). This failure prevents interference among export processes. To allow an invocation of **sync_export_list** to retry an export, use the **–iterate** and **–wait** options.

To display informational messages, specify the **–trace** option on the command line.

To display all debugging print statements, set the **TRACE_SUBSYS** environment variable to the value **sync_export_list**.

sync_export_list creates a log file during execution. This log file is deleted unless **sync_export_list** fails or you use **–trace** or set **TRACE_SUBSYS**.

By default, the log files are stored in the **/var/adm/atria/log/sync_logs** directory on UNIX and the *ccase-home-dir***var****log** directory on Windows. The name of a log file includes the process ID of the **sync_export_list** command and the time (in UTC format) at which you ran the command.

On UNIX, the Weekly Log Scrubbing job installed with ClearCase deletes **send*** and **recv*** log files in **/var/adm/atria/log/sync_logs** that are more than 14 days old.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

With **–poll**, you must have write and execute permissions on the directory containing the shipping orders, and on UNIX, you must own the shipping order files or be **root**.

Locks: An error occurs if one or more of these objects are locked: VOB.

Mastership: No mastership restrictions.

OPTIONS AND ARGUMENTS

–h·elp
Prints help on command options.

–c·ompress
Compresses update packets using Gzip compression.

–lo·gdir *log-directory*
Writes log file to *log-directory*. You must have write access to *log-directory*.

–f·ship | –sh·ip
By default, **sync_export_list** ships packets immediately (**–fship**). To store packets in the shipping bay, specify **–ship**.

–lockwait *minutes*
Number of minutes for the script to keep retrying to write to the VOB, if the VOB is locked.

–q·uiet *mode*
Suppresses messages sent to STDOUT. *mode* can have the following values:

0	(default) Prints errors, warnings, and informational messages
1	Prints errors and warnings
2	Suppresses all messages

–wo·rkdir *directory*
Writes temporary files to *directory*. *directory* must exist and be writable by the user who enters the **sync_export_list** command.

–m·axsize *max-packet-size*

Maximum size for a physical packet, expressed as a number followed by a single letter. For example:

500k	500 kilobytes
20m	20 megabytes
1.5g	1.5 gigabytes

If you do not specify **–maxsize**, **sync_export_list** uses the value specified in the **shipping.conf** file (UNIX) or **MultiSite Control Panel** (Windows). To specify no size limit, use **–maxsize 0**.

–sc·lass *storage-class*

Uses the shipping parameters associated with *storage-class*. If you do not specify **–sclass**, **sync_export_list** uses the parameters for the default storage class. You can create or modify storage classes in the **shipping.conf** file on UNIX or the **MultiSite Control Panel** on Windows.

–u·pdate

For each current replica, queries the sibling replicas for their actual states and updates the current replica's epoch table accordingly, then generates update packets. The sites must have IP connections.

–li·mit *num-packets*

Limits the number of packets **syncreplica** generates. If you also specify **–maxsize**, each packet is no larger than *max-packet-size*; otherwise, each packet is no larger than the value specified in the **shipping.conf** file (UNIX) or **MultiSite Control Panel** (Windows). Use this option when the disk space for your shipping bay or staging area is limited, or when you are creating packets to be placed on magnetic tape (UNIX) or diskettes.

–t·race

Lists command-line options you specified, displays commands as they are executed, displays a success or failure message, and forces **sync_export_list** to keep its log file.

–p·oll

Executes **shipping_server –poll** before exporting any data. If you also specify **–sclass**, **shipping_server –poll** processes only the shipping orders for the specified storage class.

–i·terate *num-tries* **–wa·it** *num-seconds*

Maximum number of tries to make all exports complete successfully, and the number of seconds to wait between tries. By default, **sync_export_list** does not retry failed exports (**–iterate 1**). If you specify **–iterate** without **–wait**, **sync_export_list** waits 30 seconds between tries.

–a·ll

Generates update packets from all replicas on the current host to all sibling replicas in their respective VOB families.

–r·eplicas *replica-list*

Generates update packets for the replicas you specify in *replica-list*. You can specify *replica-list* in any of the following forms:

replica-name@VOB-tag	Generates a packet for a replica
replica-name@VOB-tag,*replica-name*,*replica-name*,...	Generates packets for two or more replicas in a VOB family
VOB-tag	Generates update packets for all sibling replicas in a VOB family

Examples:

rep1@/vobs/dev	*(generate an update packet for a single replica)*
"rep1@\dev,rep2,rep3"	*(generate update packets for multiple replicas in a VOB family)*
\tromba	*(generate update packets for all replicas in a family)*

You can specify only one VOB family with **–replicas**. To specify multiple VOB families, use multiple **replicas:** lines in a *script-file*. You must specify at least one replica, either on the command line, or in a *script-file*.

script-file

Path to file containing directives for **sync_export_list**. You must specify this argument last on the command line. You can include the following directives in *script-file*:

compress **nocompress**	Compresses or does not compress packet.
fship	Ships packets immediately.
ship	Stores packets in shipping bay.
maxsize:*max-packet-size*	Sets maximum packet size.
sclass:*storage-class*	Sets a different storage class. To unset the storage class, do not specify a *storage-class* value.
update **noupdate**	Controls whether epoch table is updated before export.
limit:*num-packets*	Sets maximum number of packets to generate per replica.

lockwait:*minutes*	Number of minutes to wait for VOB locks.
replicas:*replica-list*	Exports packets from specified replicas. Specify *replica-list* as described in the **–replicas** option.

sync_export_list processes all directives in the order listed in *script-file*. Rules for directives:

- You can include multiple **replicas** directives in *script-file*.
- Each **replicas** directive can have different shipping directives (a shipping directive is any directive except **replicas**).
- Shipping directives must precede the **replicas** directive to which they apply.
- A shipping directive remains in effect for all subsequent **replicas** directives unless you override it.
- **sync_export_list** creates and exports packets for replicas specified on the command line, and then creates and exports packets for replicas specified in the script file.

For example, in the following script file the directives **sclass:daily** and **limit:10** apply to both **replicas** directives.

```
compress
ship
maxsize:2g
sclass:daily
update
limit:10
replicas:rep1@\myvob
nocompress
fship
maxsize:1g
noupdate
replicas:rep2@\myvob,rep3
```

EXAMPLES

- Send update packets from all replicas on the host to all their siblings.

  /usr/atria/config/scheduler/tasks/sync_export_list –all

  ```
  SUCCESSFUL COMPLETION: log file removed.
  ```

- Generate update packets for replicas in the VOB family **/vobs/dev**. Store the packets in the shipping bay, limit the size of the packets to 500KB, and display messages during processing.

  /usr/atria/config/scheduler/tasks/sync_export_list –ship –maxsize 500k –trace \ –replicas /vobs/dev

  ```
  command options specified or defaulted:
     compress: 0
     logdir:
     storage-class:
     workdir:
     maxpacket: 500k
     limit: 0
     all: 0
     fship: 0
     ship: 1
     poll: 0
     lockwait: 0 minutes
     retries: 1 times, wait 30 seconds
     script:
  CMD: bin/cleartool lsvob /vobs/dev > /dev/null
     vob: /vobs/dev
     replicas: bangalore buenosaires
  CMD: bin/multitool syncreplica -export -maxsize 500k -ship
  replica:bangalore@/vobs/dev >&2
  CMD: bin/multitool syncreplica -export -maxsize 500k -ship
  replica:buenosaires@/vobs/dev >&2


  SUCCESSFUL COMPLETION: see log file at:
  "/var/adm/atria/log/sync_logs/send-000815-183301Z-6043_log".
  ```

- Create a script file for the VOB families **\tests** and **\dev**. Create a job that runs **sync_export_list** every night at 1:00 A.M.

  Script file:

  ```
  compress
  fship
  sclass:tests
  noupdate
  replicas:sanfran_hub@\tests,sydney
  sclass:dev
  update
  replicas:\dev
  ```

Job definition:

```
Job.Begin
   Job.Id: 25
   Job.Name: "Sync Export tests dev"
   Job.Description.Begin:
Every midnight, export update packets to replicas in VOB families \tests
and \dev.
   Job.Description.End:
   Job.Schedule.Daily.Frequency: 1
   Job.Schedule.FirstStartTime: 01:00:00
   Job.DeleteWhenCompleted: FALSE
   Job.Task: 13
   Job.Args: -quiet 1 \\shinjuku\scripts\sync_export_tests_dev
Job.End
```

FILES

UNIX

/var/adm/atria/log/sync_logs
/var/adm/atria/config/shipping.conf
*ccase-home-dir***/config/scheduler/multisite.schedule**

Windows

*ccase-home-dir***\var\log**

SEE ALSO

mkorder, **MultiSite Control Panel**, **shipping.conf**, **shipping_server**, **sync_receive**, **syncreplica**

sync_receive

Imports update packets

APPLICABILITY

Product	Command type
MultiSite	MultiSite command

Platform
UNIX
Windows

SYNOPSIS

- Import update packets:

  sync_receive [**–v·ob** *pattern*] [**–wo·rkdir** *directory*] [**–lo·gdir** *log-directory*]
  [**–lockwait** *minutes*] [**–t·race**] [**–q·uiet** *mode*] [**–d·ata** [*packet-file-pname* | *dir*]]
  [**–a·ctual** *shipping-order-pname*] [**–s·class** *storage-class*] [**–o·rigin** *hostname*]

- Print help on command options:

  sync_receive –h·elp

On UNIX, **sync_receive** is located in *ccase-home-dir*/**config/scheduler/tasks**. On Windows, **sync_receive** is located in *ccase-home-dir***config\scheduler\tasks**.

DESCRIPTION

sync_receive imports update packets in the local host's incoming storage bays. You can run **sync_receive** from the command line, or run it with the **schedule** command (see the **schedule** reference page in the *Command Reference*). For information about using **sync_receive** as a receipt handler, see the **shipping.conf** and **MultiSite Control Panel** reference pages.

If files in the incoming shipping bays have names ending with `.gz`, **sync_receive** uncompresses the files, determines whether they are packets, and then imports the packets.

RETRYING SYNCHRONIZATION WHEN THE VOB IS LOCKED

By default, synchronization imports fail if the VOB is locked. To allow **sync_receive** to retry an import when it encounters a lock, use the **–lockwait** option, which specifies the amount of time (in minutes) for **sync_receive** to keep trying to write to the VOB. During that time, **sync_receive**

retries the write operation every minute. If the time elapses and the VOB is still locked, **sync_receive** exits with an error.

The **–lockwait** option sets the CLEARCASE_VOBLOCKWAIT environment variable in the script's environment. If **–lockwait** is not used, **sync_receive** ignores CLEARCASE_VOBLOCKWAIT if it is set outside the script's environment.

NOTE: **sync_receive** waits only if it detects the lock before it starts processing oplogs. If an administrator locks the VOB during oplog processing, **sync_receive** exits with an error.

CONFIGURATION FILE

You can modify the behavior of the **sync_receive** script by creating a file named **MSimport_export.conf** and setting values in it. On UNIX, create the file in the directory **/var/adm/atria/config**. On Windows, create the file in the directory *ccase-home-dir***var****config**.

The file can include the following import settings:

disable_import_locking = 1

Disables use of the import lockfile, allowing multiple imports to a single replica to run simultaneously. Setting the value to 0 (default) enables use of the lockfile.

NOTE: By default, **sync_receive** fails if there is another **sync_receive** process importing a packet into the same replica. This failure prevents interference among import processes. Disabling import locking may cause import failures due to collisions. We recommend that you leave locking enabled unless there is a large amount of lockfile contention.

This setting and the **–lockwait** option are not related. This setting configures use of the lock created by the **sync_receive** process to prevent interference among import processes, and the **–lockwait** option handles ClearCase VOB locks.

proactive_receipt_handler = 1

Causes an active receipt handler to look for other packets that can be imported and attempt to import them. By default, a receipt handler imports only the packet for which it was invoked. Under high load conditions, or when packet have been split because of maximum size restrictions, packets may arrive before a preceding packet has been completely processed. Enabling proactive mode causes the receipt handler to import packets that may otherwise be stranded due to premature or out-of-order delivery.

TROUBLESHOOTING

To display informational messages, specify the **–trace** option on the command line.

To display all debugging print statements, set the TRACE_SUBSYS environment variable to the value **sync_receive**.

sync_receive creates a log file during execution. This log file is deleted unless **sync_receive** fails or you use **–trace** or TRACE_SUBSYS.

By default, the log files are stored in the **/var/adm/atria/log/sync_logs** directory (UNIX) or the *ccase-home-dir***var****log** directory (Windows). The name of a log file is based on the process ID of the **sync_export_list** command and the time at which you ran the command.

On UNIX, the Weekly Log Scrubbing job installed with ClearCase deletes **send*** and **recv*** log files in **/var/adm/atria/log/sync_logs** that are more than 14 days old.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB.

Mastership: No mastership restrictions.

OPTIONS AND ARGUMENTS

–h·elp
Prints help on command options.

–v·ob *pattern*
VOBs to which update packets are applied. By default, **sync_receive** applies packets to all VOBs listed in the packet. Specify *pattern* as a VOB-tag or as a string that can match multiple VOB names. You cannot include wildcard characters in *pattern*. For example:

`-vob /vobs/dev`	*(apply packets to /vobs/dev and any VOB whose tag contains '/vobs/dev')*
`-vob dev`	*(apply packets to any VOB whose tag contains the string 'dev')*

–wo·rkdir *directory*
Writes temporary files to *directory*. *directory* must exist and be writable by the user who enters the **sync_receive** command.

–lo·gdir *log-directory*
Writes log file to *log-directory*. You must have write access to *log-directory*. By default, log files are stored in the **/var/adm/atria/log/sync_logs** directory on UNIX and the *ccase-home-dir***var****log** directory on Windows.

–lockwait *minutes*
Number of minutes for the script to keep retrying to write to a locked VOB.

–t·race
Lists command-line options you specified, displays commands as they are executed, displays a success or failure message, and forces **sync_receive** to keep its log file.

–q·uiet *mode*

Suppresses messages sent to STDOUT. *mode* can have the following values:

0	(default when **sync_receive** is used on the command line) Prints errors, warnings, and informational messages
1	(default when **sync_receive** is used as a receipt handler) Prints errors and warnings
2	Suppresses all messages

When **sync_receive** is invoked as a receipt handler, *mode* is set to 1.

When **sync_receive** is invoked as a receipt handler, the following parameters are passed in automatically. You can use **–sclass**, **–data**, and **–actual** from the command line.

–s·class *storage-class*

Imports packets in the incoming bays associated with *storage-class*. If *storage-class* does not have incoming bays or you do not specify **–sclass**, **sync_receive** imports packets from the shipping bay for the default storage class.You can create and modify storage classes in the **shipping.conf** file on UNIX or the **MultiSite Control Panel** on Windows.

–d·ata [*packet-file-pname* | *dir*]

Full pathname of an update packet or a storage bay. To import only a specific packet, use **–data** *file*. To import all packets in a bay, use **–data** *dir*. You can use **–data** with **–vob** to import packets to specific VOBs. This parameter is used only when the packet is destined for replicas on the current host.

–a·ctual *shipping-order-pname*

Location of the shipping order; used only when the packet is destined for another host.

If a packet is destined for both the local host and another host, both the **–data** and **–actual** parameters are used. The packet is imported at the replica on the local host, and then forwarded to its next destination.

NOTE: This option is not related to the **–actual** option for **chepoch** and **lsepoch**.

–o·rigin *hostname*

Originating host.

EXAMPLES

- Import packets in the incoming storage bays for the **daily** storage class.

  /usr/atria/config/scheduler/tasks/sync_receive –sclass daily

- Import a specific packet and apply it to all VOBs whose tags include the pattern **lib**. The lines are broken for readability. You must enter the command on a single physical line.

"c:\Program Files\Rational\ClearCase\config\scheduler\tasks\sync_receive.bat" –vob lib –d "c:\Program Files\Rational\ClearCase\var\shipping\daily\incoming\sync_orig_09-Dec-98.18.17.54_6587_1"

- On UNIX, specify **sync_receive** as the receipt handler for the **daily** storage class.

  cp /usr/atria/config/scheduler/tasks/sync_receive* /var/adm/atria/scheduler/tasks

  Edit the **shipping.conf** file and add a receipt handler entry:

  RECEIPT-HANDLER daily /var/adm/atria/scheduler/tasks/sync_receive

- On Windows, specify **sync_receive** as the receipt handler for the **daily** storage class.

  a. Copy the script into a directory outside the ClearCase installation area. For example:

copy "c:\Program Files\Rational\ClearCase\config\scheduler\tasks\sync_receive.bat" c:\scripts

  b. Edit the script as appropriate.

  c. In the MultiSite Control Panel, select the daily class in the **Storage Class** list.

  d. Click **Modify Class**.

  e. In the **Receipt Handler Path** box, enter the path to the script. For example:

  c:\scripts\sync_receive.bat

  f. Click **OK**.

FILES

UNIX

/var/adm/atria/log/sync_logs
/var/adm/atria/config/shipping.conf
ccase-home-dir/**config/scheduler/multisite.schedule**

Windows

*ccase-home-dir***var\log**

SEE ALSO

mkorder, **MultiSite Control Panel**, **shipping.conf**, **shipping_server**, **sync_export_list**, **syncreplica**

syncreplica

Generates or applies update packets

APPLICABILITY

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

SYNOPSIS

- Export an update packet:

  sync·replica –exp·ort [**–max·size** *max-packet-size* [**–lim·it** *num-packets*]]
  [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
  {
  { **–sh·ip** | **–fsh·ip** } [**–scl·ass** *storage-class*]
  [**–pex·pire** *date*] [**–not·ify** *e-mail-addr*]
  | **–tape** *raw-device-pname*
  | **–out** *packet-file-pname*
  }
  replica-selector ...

  NOTE: The **–tape** option is valid only on UNIX.

- Import an update packet:

  sync·replica –imp·ort [**–invob** *VOB-selector*]
  [**–c·omment** *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**]
  { **–rec·eive** [**–scl·ass** *storage-class*]
  | **–tape** *raw-device-pname*
  | { *packet-file-pname* | *staging-area-pname* } ...
  }

  NOTE: The **–tape** option is valid only on UNIX.

DESCRIPTION

Synchronization of an existing VOB replica with one or more replicas at other sites is a two-phase process:

1. At one site, a **syncreplica –export** command creates an update packet that contains changes that have occurred in the VOB replica at that site (and perhaps other replicas, as well).
2. At another site, a **syncreplica –import** command applies the changes in the update packet to its replica of the same VOB.

Step #2 occurs at all sites that receive the packet.

Contents of an update packet:

- All changes that have occurred in the current VOB replica since the last update generated for the destination replicas. (Changes already sent to the destination replicas are excluded from the packet).
- Changes that have occurred in other replicas, which the current replica has received in previous update packets from those replicas, but has not already passed on to the destination replicas.

In all cases, **syncreplica –export** creates a single logical update packet for use at all the specified destinations; the packet can be used to update *only* those particular replicas.

NOTES ON THE EXPORT PHASE

MultiSite is designed for efficient updating of replicas. **syncreplica –export** attempts to exclude from an update packet operations that have been sent previously. (However, there is no harm in sending an operation multiple times to the same replica; the first operation is imported and subsequent identical operations are ignored.)

The VOB replica is not locked during the export phase; in fact, the **syncreplica –export** command fails if the VOB is locked. Therefore, you must not schedule synchronizations during VOB backups (when the VOB must be locked). See also *RETRYING SYNCHRONIZATION WHEN THE VOB IS LOCKED* on page 313.

Specifying a Directory for Temporary Files

syncreplica –export stores temporary files in the directory specified by the **TMPDIR** environment variable on UNIX and the **TMP** environment variable on Windows. If you use the **sync_export_list** script to export update packets, you can use the **–workdir** option to specify the directory.

NOTES ON THE IMPORT PHASE

An update packet is applied to the appropriate replica on the host on which you import it, unless you restrict processing with the **–invob** argument. **syncreplica** consults the VOB registry in the

current region to determine the locations of these replicas' storage directories. Thus, you do not have to specify particular replicas or storage locations.

The import process applies update packets in the correct order. Therefore, you can specify packets in any order on the command line.

The VOB replica is not locked during the import phase. Synchronization fails if the VOB is locked. See also *RETRYING SYNCHRONIZATION WHEN THE VOB IS LOCKED* on page 313.

Specifying a Directory for Temporary Files

syncreplica –import stores temporary files in the directory specified by the **TMPDIR** environment variable on UNIX and the **TMP** environment variable on Windows. If you use the **sync_receive** script to import update packets, you can use the **–workdir** option to specify the directory.

Skipping Packets

syncreplica –import refuses to process an update packet in the following situations:

- The update packet contains changes that depend on other changes that have not yet been applied to this replica. This usually means that an update packet destined for this replica has not been sent or was lost during transport.
- Problems were encountered processing an earlier physical packet in a multiple-part logical packet.

In any of these cases, **syncreplica –import** displays an explanatory message.

Update Failures / Replaying Packets

In some cases, **syncreplica –import** begins to apply operations to a replica, but fails with an error message. For example, another process may have locked the VOB, causing the import to fail. After the VOB is unlocked, you can run **syncreplica –import** to process the entire update packet again.

There is no harm in importing update packets that have already been processed successfully; the same change will not be made twice. Thus, even importing an entire update packet multiple times causes no error and does no harm.

For more information about update failures, see Chapter 10, *Troubleshooting MultiSite Operations*.

Deletion of Update Packets

If a single invocation of **syncreplica –import** applies a packet successfully to all target replicas on the host, the update packet is deleted when the command completes its work. If the packet is processed with multiple **syncreplica –import –invob** commands, it is not deleted.

Ownership Preservation

If a VOB replica is ownership-preserving, **syncreplica –import** maintains the consistency of ownership and permissions information for elements mastered by the VOB family's ownership-preserving replicas. For each such element, an error occurs if the element's group is not on the group list of the importing replica (on UNIX) or is not the same as the group of the importing replica (on Windows).

If a VOB replica is not ownership-preserving, changes to ownership and permissions of existing elements are ignored during import. New elements are assigned to the owner of the VOB at the current site, and the group of all new elements is the primary group of the owner of the VOB. (This is true even if the **root** user or a member of the ClearCase administrators group imports the packet.) Permissions set when the element is created are preserved, but subsequent permissions changes are ignored. Ownership and permissions changes made at non-ownership-preserving replicas are not propagated to other replicas.

Storage Pools

Data containers from the update packets are placed in storage pools according to the standard element assignments. If the pool assignment for a new element cannot be determined, the element is assigned to the VOB's default source pool.

Trigger Firing

ClearCase triggers do not fire in response to changes made during packet import.

Handling Naming Conflicts

syncreplica resolves naming conflicts among type objects created at different replicas. For more information, see *Conflict Resolution* on page 21.

Delayed View Updates

syncreplica does not inform any views (not even the view from which you enter the command) of the updates to replicas. All active views see updates within a few seconds, through their normal VOB-polling routines. You can force a view to recognize VOB updates by entering a **cleartool setcs –current** command.

RETRYING SYNCHRONIZATION WHEN THE VOB IS LOCKED

By default, synchronization exports and imports fail if the VOB is locked. To allow **syncreplica** to retry a synchronization when it encounters a lock, set the CLEARCASE_VOBLOCKWAIT environment variable to the amount of time (in minutes) for **syncreplica** to keep trying to write to the VOB. During that time, **syncreplica** retries the write operation every minute. If the time elapses and the VOB is still locked, **syncreplica** exits with an error.

NOTE: The **syncreplica** command waits only if it detects the lock before it starts processing oplogs. If an administrator locks the VOB during oplog processing, **syncreplica** exits with an error.

RESTRICTIONS

Identities: You must have one of the following identities:

- VOB owner
- **root** (UNIX)
- Member of the ClearCase administrators group (Windows)

Locks: An error occurs if one or more of these objects are locked: VOB.

Mastership: No mastership restrictions.

OPTIONS AND ARGUMENTS — EXPORT PHASE

The following sections describe the options and arguments for use with **syncreplica –export**.

SPECIFYING THE UPDATE PACKET SIZE. *Default:* When you do not specify **–maxsize**, the default packet size depends on the shipping method you use:

- Packets created with **–ship** or **–fship** are no larger than the maximum packet size specified in the **shipping.conf** file (UNIX) or the **MultiSite Control Panel** (Windows).
- Packets created with **–out** are no larger than 2 GB.
- Packets created with **–tape** have no default size limit.

–max·size *max-packet-size* [**–lim·it** *num-packets*]

The maximum size for a physical packet, expressed as a number followed by a single letter. For example:

500k	500 kilobytes
20m	20 megabytes
1.5g	1.5 gigabytes

The **–limit** option limits the number of packets **syncreplica** generates; each packet is no larger than *max-packet-size*. Use this option when the disk space for your shipping bay or staging area is limited.

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**

Overrides the default with the specified comment option.

DISPOSITION OF THE UPDATE PACKET. *Default:* None. You must specify how the update packets created by **syncreplica –export** are to be stored and/or transmitted to other sites.

–shi·p
–fsh·ip

Stores the update packet in one or more files in a store-and-forward storage bay; **syncreplica** creates a separate shipping order for each physical packet, indicating how and where it is to be delivered. The destinations are the host names associated in the VOB database with the *replica-name* arguments. (Replica-name/host-name associations are created with **mkreplica –export** and can be changed with **chreplica**.)

Using **–fship** (force ship) invokes the **shipping_server** to send the update packet immediately. Using **–ship** does not invoke this server. To run **shipping_server** to send packets in storage bays, schedule **sync_export_list –poll** with the **schedule** command. (See the **schedule** reference page in the *Command Reference*.)

–scl·ass *class-name*

Specifies the storage class of the packet and shipping order. **syncreplica** looks up the storage class in the **shipping.conf** file on UNIX or the **MultiSite Control Panel** on Windows to determine the location of the storage bay to use.

If you omit this option, **syncreplica** places the packet in the storage bay location specified for the **–default** class in the **shipping.conf** file or **MultiSite Control Panel**. By default, this location is **/var/adm/atria/shipping/ms_ship** on UNIX and *ccase-home-dir***\var\shipping\ms_ship** on Windows.

–tap·e *raw-device-pname* (UNIX)

Writes the update packets to the specified tape device, which must be local to the host on which you enter the **syncreplica** command. You are prompted to load a separate tape for each physical packet. Use the **–maxsize** option to ensure that **syncreplica** does not exceed the capacity of the tapes you are using. Only one physical packet can be placed on each tape, regardless of packet size.

CAUTION: Be sure to deliver a packet created with **–out** or **–tape** to its specified destinations promptly. If a replica has not yet received and applied this packet, it may not accept any subsequently generated packets from your site until the first packet is received and processed.

–out *packet-file-pname*

Places the first update packet in file *packet-file-pname*. Additional physical packets, if any, are placed in files named *packet-file-pname_2*, *packet-file-pname_3*, and so on.

The update packets are not delivered automatically; use an appropriate mechanism (electronic mail, **ftp**, postal service, and so on) to deliver them.

You can create a packet using **–out**, and deliver it using the store-and-forward facility. See the **mkorder** reference page.

HANDLING PACKET-DELIVERY FAILURES. *Default:* If a packet cannot be delivered, it is sent through the store-and-forward facility back to the administrator at the site of the originating replica. A mail message is sent to the store-and-forward administrator. This occurs after repeated attempts to deliver the packet have all failed, and the allotted time has expired; it can also occur when the destination host is unknown or a data file does not exist. The store-and-forward configuration settings specify the expiration period and the e-mail address of the administrator.

–pex·pire *date-time*

Specifies the time at which the store-and-forward facility stops attempting to deliver the packet and generates a failure mail message instead.

UNIX: This option overrides the storage class's **EXPIRATION** specification in the store-and-forward configuration file. See the **shipping.conf** reference page for a description of this specification, and of delivery retries in general.

Windows: This option overrides the storage class's Packet Expiration specification in the MultiSite Control Panel. See the **MultiSite Control Panel** reference page for a description of this specification, and of delivery retries in general.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**
where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** | **yesterday** | **Sunday** | ... | **Saturday** | **Sun** | ... | **Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... | **December** | **Jan** |... | **Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–not·ify *e-mail-address*
The delivery-failure message is sent to the specified e-mail address.

If a failure occurs on a Windows host that does not have e-mail notification enabled, a message appears in the Windows Event Viewer. The message includes the *e-mail-address* value specified with this option and a note requesting that this user be informed of the status of the operation. For information about enabling e-mail notification, see the **MultiSite Control Panel** reference page.

SPECIFYING THE DESTINATION REPLICAS. *Default:* None.

replica-selector ...
Prepares an update packet to be sent to the specified replicas, which must be in the same VOB family. Specify *replica-selector* in the form [**replica:**]*replica-name*[@*vob-selector*]

replica-name Name of the replica (displayed with **lsreplica**)

vob-selector VOB family of the replica; can be omitted if the current working directory is within the VOB.

Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

OPTIONS AND ARGUMENTS — IMPORT PHASE

The following sections describe the options and arguments for use with **syncreplica –import**.

RESTRICTING THE UPDATE TO A PARTICULAR VOB. *Default:* Updates all replicas that are on the current host and are specified in the update packets. With **–tape**, you must specify a VOB replica to be updated.

–invob *vob-selector*
Updates the replica in the VOB family specified by *vob-selector*; all other replicas specified in the update packets are ignored. Specify *vob-selector* in the form [**vob:**]*pname-in-vob*

pname-in-vob Pathname of the VOB-tag (whether or not the VOB is mounted) or of any file-system object within the VOB (if the VOB is mounted)

EVENT RECORDS AND COMMENTS. *Default:* Creates one or more event records, with commenting controlled by the standard ClearCase user profile (default: **–nc**). See *EVENT RECORDS AND COMMENTS* in the **multitool** reference page. To edit a comment, use **cleartool chevent**.

–c·omment *comment* | **–cfi·le** *comment-file-pname* | **–cq·uery** | **–cqe·ach** | **–nc·omment**
Overrides the default with the specified comment option.

SPECIFYING THE LOCATION OF THE UPDATE PACKETS. *Default:* None.

–rec·eive [**–scl·ass** *storage-class*]
Scans one or more of the current host's storage bays. Any unprocessed update packets intended for this host are applied to the appropriate replicas on the host. Using the **–sclass** option restricts processing to the storage bays of the specified storage class.

If **syncreplica** finds any replica-creation packets, it sends mail to the store-and-forward administrator. (If the current host is a Windows host and there is no valid host specified in the **SMTP Host** box in the ClearCase Control Panel, a message appears in the Windows Event Viewer.) Use **mkreplica** to import these replica-creation packets.

–tap·e *raw-device-pname* (UNIX)
Reads a single packet from the tape device, and applies it to the replica of the VOB specified with **–invob**. The tape device must be local to the importing host.

packet-file-pname | *staging-area-pname* ...
Processes each *packet-file-pname* as an update packet. For each *staging-area-pname* specified, locates all previously unprocessed update packets in the directory and applies them to the appropriate replicas.

EXAMPLES

Exports

- Generate an update packet to be sent to replica **boston_hub**. Store the packet in a file in directory **c:\tmp**.

  multitool syncreplica –export –out c:\tmp\boston_hub_packet1 boston_hub@\dev
  ```
  Generating synchronization packet c:\tmp\boston_hub_packet1
  ```

- Similar to preceding example, but place the packet file in a storage bay, for shipping at some later time by the store-and-forward facility.

  multitool syncreplica –export –ship boston_hub@\dev
  ```
  Generating synchronization packet c:\Program Files\Rational\ClearCase\var
  \shipping\ms_ship\outgoing\sync_bangalore_19-May-99.09.33.02_3447_1
   - shipping order file is c:\Program
  Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\sh_o_sync_bangalore
  _19-May-99.09.33.02_3447_1
  ```

- Similar to preceding example, but ship the packet immediately.

  multitool syncreplica –export –fship boston_hub@\vob2
  ```
  Generating synchronization packet c:\Program Files\Rational\ClearCase\var
  \shipping\ms_ship\outgoing\sync_bangalore_19-May-99.09.33.02_3447_1
   - shipping order file is c:\Program
  Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\sh_o_sync_bangalore
  _19-May-99.09.33.02_3447_1
  Attempting to forward/deliver generated packets...
   -- Forwarded/delivered packet c:\Program Files\Rational\ClearCase\var
  \shipping\ms_ship\sync_bangalore_19-May-99.09.33.02_3447_1
  ```

Imports

- Process an incoming update packet in directory **/usr/tmp**.

  multitool syncreplica –import /usr/tmp/boston_hub_packet1
  ```
  Applied sync. packet /usr/tmp/boston_hub_packet1 to VOB
  /net/minuteman/vobstg/dev.vbs
  ```

- Process all incoming update packets in the current host's storage bays.

  multitool syncreplica –import –receive
  ```
  Applied sync. packet c:\Program Files\Rational\ClearCase\var
  \shipping\ms_ship\incoming\sync_boston_hub_19-May-99.09.45.01_7634_1
  to VOB \\ramohalli\vobs\dev.vbs
  ```

SEE ALSO

mkorder, **mkreplica**, **MultiSite Control Panel**, **shipping.conf**, **sync_export_list**
Chapter 10, *Troubleshooting MultiSite Operations*

Index

/usr/atria/bin directory 53
/usr/atria/config/services/shipping.conf file, *See* shipping.conf file
/usr/atria/etc directory 53
/var/adm/atria/log directory 164
/var/adm/atria/log/shipping_server_log file 164

A

ACLs
mastership requests 277
storage bays 265
administration
backup requirements 51
disk space for storage bays 34
list of responsibilities 49
scrubbing 47
storage registries 50
albd_server, control of ports used 101
apropos command 211
asterisks in packet listings 237
***attache-home-dir* directory** xix

B

backup
incremental 201
nonreplicated objects 200
replica as mechanism for 199
requirements 51
synchronization schedule 88
bays, *See* return bays; storage bays
bidirectional synchronization
about 42
feature levels 82
branch mastership
See also mastership
about 8
assigning when creating elements 37, 124
conditions for enabling requests 144
creating type objects 15
default vs. explicit 13
displaying request settings 124, 150
how used 10
implementation planning issues 145
models for serial development 20
planning scenario 69
removing explicit 133
request mechanism, setup procedure 146
request mechanisms 141
request procedure 141
scope 9
serial development scenario 157
strategy for branching and merging 36
transfer models 36
transfer procedure 132
branch types, transferring mastership of 127
branches
requesting mastership of 276

C

***ccase-home-dir* directory** xix
ccase-home-dir\bin directory 53
ccase-home-dir\config\scheduler\tasks directory 53
ccase-home-dir\var\log directory 164
ccase-home-dir\var\log\shipping_server_log file 164
chepoch command 213
chreplica command 224
ClearCase commands, use with replicas 60
ClearCase scheduler, synchronization jobs 104
CLEARCASE_MAX_PORT environment variable 101
CLEARCASE_MIN_PORT environment variable 101
.clearcase_profile file 271
cleartool and multitool commands 55
commands for MultiSite
about 53
ClearCase 60
multitool 54
non-multitool 59
when view context is useful 58
connectivity property
changing 224

conventions, typographical xix
creating replicas
about 65
command for 249
common problems 165
export procedure 71
import procedure 74
in mixed environment 77
scenario 68
when to schedule 66
with store-and-forward facility 66

D

directories
/usr/atria/bin 53
/usr/atria/etc 53
ccase-home-dir\bin 53
ccase-home-dir\config\scheduler\tasks 53
changing in replicas 61
disk space
for backup replica 199
replica-creation directory 71
storage bays 34
documentation
MultiSite online help description xx

E

element types, deleting 61
elements
assignment of mastership 124
preservation of ownership 4
transfer of mastership 131
e-mail and firewalls 96
encryption of update packets 100
environment variables 101
epoch number matrix
about 27
listing contents of 28, 229
zeros in 29
epoch numbers
about 24
changing, commands for 213, 272
changing, methods for 169, 182
checking 227
gap detected during packet creation 170
gaps in 168
role in updates 26
epoch_watchdog command 227
error messages
See also troubleshooting
`Gap in oplog detected for replica` 170
`Gap in oplog entries` 168
`Replica already exists` 165
transport operations, list of 172
event records
about 24
comments in 270
export operations
automating for synchronization 105
creating update packets 297
element is checked out 170
gap in epoch numbers 170
packets accumulate in storage bay 171
replica creation 66, 71
replica-creation packets, recovering lost 182
resending lost packets 272
synchronization problems 167
synchronization procedure, manual 102
update packet delivery patterns 86
export_sync records, scrubbing 49

F

feature levels
about 79
displaying 83
raising for replica 80
raising for VOB family 82
requests for branch mastership 144
firewalls
shipping_server on 98
synchronization and 96
ftp and firewalls 97

H

help xx
host name
changing for a replica 224
host name of replica, changing 113
hyperlink types, shared 15

I

import operations
assumption of success 90
automating for synchronization 107–108
common synchronization problems 175
conflicts in registry 166
corrupted packet symptoms 178
failure of and replica replacement 192
failures, possible causes 181
lost packets 176, 182

replica creation 74
synchronization command 305
synchronization procedure, manual 103
when to restart 178
installation and licensing 34
interoperability 203

L

licenses needed for ClearCase and MultiSite 34
local-area networks, interoperability 203
log files, locations of 164
lsepoch command 229
lspacket command 237

M

man command 53–54
master replica, setting access control for 276
mastership
See also branch mastership
about 7
changing 217
creating type objects 137
displaying request settings 124
elements, transferring 131
fixing accidental change in 137
management of 121
objects in removed replicas 119
of replica object 8
request failed 151
restrictions for VOB objects 17
transferring 126
transferring, replica removal 135
troubleshooting for type objects 186
type objects 14, 127
VOBs, transferring 130
mkorder command 244
mkreplica command 249
MultiSite Control Panel 92, 263
multitool commands
about 54
summary 55
syntax for 268

O

object selectors for multitool commands 269
objects
See type objects; VOB objects
online help xx
oplogs (operation logs)
about 24
gaps in epoch numbers 168
scrubbing 47
ownership-preserving replicas
about 4
behavior of syncreplica -import 313
changing properties of 114, 224
creating 40, 250
requirements 5
troubleshooting on UNIX 185
UNIX and Windows interoperability 205

P

packets
See also replica-creation packets; update packets
about 6
listing contents of 237
logical and physical 6
processing imported 6
redelivering 264, 291
routing 267, 292
splitting logical into physical 263, 289
submitting to store-and-forward facility 92
planning issues
about 33
branch mastership 145
design documentation 33
firewalls 100
licensing 34
synchronization strategy 46
time zones and synchronization strategy 88
ports, control of for servers 101
privileges, *See* mastership

R

receipt handlers, paths 266, 292
recoverpacket command 272
replica objects
about 2
deleting 287
mastership 8
transferring mastership of 117, 128
replica-creation packets
contents and cleanup 251
how to split 67
replicas
See also creating replicas; ownership-preserving replicas; synchronizing replicas
about 2

accidental deletion, recovery 196
as backup mechanism 199
backing up 51
changing connectivity property 224
changing hosts or host names 113, 224
checking epoch number 227
displaying details of 23
displaying properties of 111
feature levels 79–80
history of changes, how tracked 25
listing 240
listing objects mastered by 232
master, of VOB and type objects, displaying 122
moving 117
multiple at one site 205
names 2
removal procedure 119
renaming 116
replacing 192
resolving name conflicts 21
restoring from backup 190–191
scrubbing oplogs 47
self-mastering 8, 117, 128
site differences 2
transferring mastership of objects in 135
UNIX and Windows interoperability 203
where mounted 50

reqmaster command, status messages 152

restorereplica command 283

return bays
See also storage bays
about 91
ACLs 265
handling packets in 175
paths 265, 290

rmreplica command 287

S

scrubbing 47

serial development
branch mastership models 20
branch mastership scenario 157

shipping orders
about 91
creating 244
expiration date, specifying 264, 291
expired 95, 175
processing 294, 297

shipping.conf file
about 92
modifying contents of 289

shipping_server
about 92, 294
error handling mechanisms 94
installing on firewall 98
log file 295
troubleshooting scenarios 171

sites
about 1
differences among 2
documentation of design 33
multiple replicas at single 205

storage bays
See also return bays
about 91
ACLs 265
disk space requirements 34
packets in 171, 176
paths 265, 290

storage classes
naming 264
use in synchronization 95

storage directories, restoring lost 190

storage registries, where mounted 50

store-and-forward facility
about 90
configuring 289
creating replicas 66
creating shipping orders 244
customizing 263
deliveries attempted 94
firewalls 98
indirect shipping routes 93
notification mechanisms 290
reliability of and packet size 67
sending files with 93
storage classes 95
submitting packets 92
use with firewalls 96

sync_export_list command 297

sync_receive command 305

synchronizing replicas
about 6, 85
assumption of success 90
automating 104
command for 310
common export problems 167
data included and excluded 3
deliveries attempted 94
delivery patterns 86
direction of, and feature levels 82
firewalls, methods for handling 96
history 113
inconsistent changes 184
indirect routes 29
manual procedure 102
planning issues 46
risks of scrubbing oplogs 49

risks of unidirectional scheme 43
role of epoch numbers 26
schedule for 87
unidirectional vs. bidirectional 42
VOB database mechanism 24
syncreplica command 310
examples 102

T

TCP applications and firewalls 98
technical support xx
time stamps, interpretation of format 6
time zones, issues for synchronization strategies 88
topology for update packets 86
transport operations
automating for synchronization 106
common problems 172
delivery failure 174
delivery mechanisms 6
firewalls 96
in mixed environment 77
indirect routes 93
invalid destinations 173
recommended methods 41
replica creation 73
shipping order expired 175
shipping_server 294
store-and-forward facility 90
synchronization procedure, manual 103
triggers
firing during synchronization 313
propagating 3
troubleshooting
about 163
accidental transfer of mastership 137
conflicts in registry entries 166
deliveries, reattempting 94
delivery failed 174
diagnostic tips 164
expired shipping order 175
export of checked-out element 170
export of update packets 167
gap in oplog entries 168
import failed 181
import failure and replica replacement 192
import problems 175
incoming packets accumulate 176
invalid destinations 173
log files 164
lost packets 182
names of type objects conflict 187
object mastership problems 186
ownership-preserving replicas 185
packet size for store-and-forward facility 67
recovery from VOB server crash 191
replica already exists 165
replica deleted 196
replica-creation problems 165
requests for mastership 151
shipping_server log files 295
shipping_server problems 171
storage registries 50
success of synchronization 90
synchronization and scrubbed oplogs 49
synchronization log files 104
tracing exported update packets 298
tracing imported update packets 306
transport problems 172
update packet creation 168
type objects
conversion of, restrictions 15
converting unshared to shared 138
creating instances 137
creating instances of shared 61
creating shared 137
creating shared and unshared 14
displaying master replica 122
displaying mastership status 138
identical names and types 21
mastership 14
mastership problems 186
renaming 187
transferring mastership 127
typographical conventions xix

U

unidirectional synchronization
about 42
feature levels 82
risks 43
update packets
automating creation of 105
automating import of 108
contents of 311
creating manually 102
deleting 312
encryption 100
error notification in mixed environments 95
storage classes 95
user profile file 271

V

var\log directory 164
var\log\shipping_server_log file 164
version information, displaying for MultiSite 268
views
data in, synchronizing 3
saving from replaced replica 195
updating with replica changes 313
use in troubleshooting 164
VOB database, mechanism for replica synchronization 24
VOB families
about 2
feature level for branch mastership request 144
feature levels 79, 82
ownership-preserving replicas in 5
preserving element ownership 4
VOB objects
displaying master replica 122
mastership restrictions 17
non-file-system 269
syntax for names 270
VOBs
transfer of mastership 130
VOBs, structure of 2
VOB-tags
assigning public 50
duplicate 167
replica names and 2