Rational Software Corporation®

Rational® ClearCase MultiSite®

Administrator's Guide

VERSION: 2003.06.00 AND LATER

PART NUMBER: 800-026169-000

UNIX/WINDOWS EDITION

Rational®
the software development company

support@rational.com
http://www.rational.com

Legal Notices

Part Number: 800-026169-000
Version Number: 2003.06.00 and later

Third Party Notices, Code, Licenses, and Acknowledgements

Contents

MultiSite Overview

Replication and Synchronization

Troubleshooting

MultiSite Reference Pages

Figures

Tables

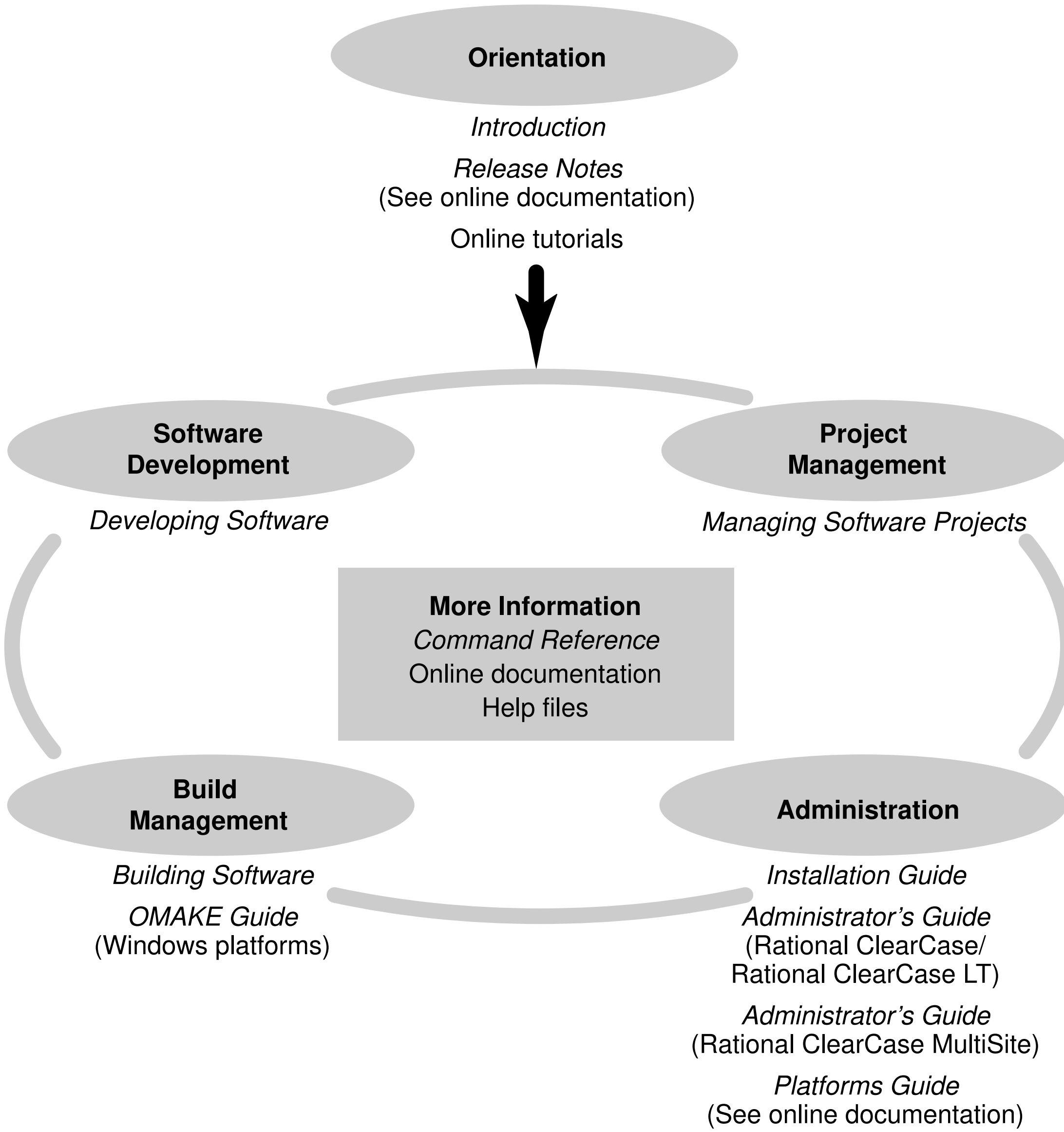
Preface

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). You can also use MultiSite as a VOB interoperation solution in a mixed UNIX and Windows network or as a VOB 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.

ClearCase Documentation Roadmap



ClearCase Integrations with Other Rational Products

Integration	Description	Where it is documented
Base ClearCase-ClearQuest	Associates change requests with versions of ClearCase elements.	ClearCase: *Developing Software* ClearCase: *Managing Software Projects* ClearQuest: *Administrator's Guide*
Base ClearCase-Apex	Allows Apex developers to store files in ClearCase.	*Installing Rational Apex* (UNIX)
Base ClearCase-ClearDDTS	Associates change requests with versions of ClearCase elements.	*ClearCase ClearDDTS Integration*
Base ClearCase-PurifyPlus	Allows developers to invoke ClearCase from PurifyPlus.	PurifyPlus Help
Base ClearCase-RequisitePro	Archives RequisitePro projects in ClearCase.	*RequisitePro User's Guide* RequisitePro Help
Base ClearCase-Rose	Stores Rose models in ClearCase.	Rose Help
Base ClearCase-Rose RealTime	Stores Rose RealTime models in ClearCase.	*Rose RealTime Toolset Guide* *Rose RealTime Guide to Team Development*
Base ClearCase-SoDA	Collects information from ClearCase and presents it in various report formats.	*Using Rational SoDA for Word* *Using Rational SoDA for Frame* SoDA Help
Base ClearCase-XDE	Stores XDE models in ClearCase	XDE Help
UCM-ClearQuest	Links UCM activities to ClearQuest records.	ClearCase: *Developing Software* ClearCase: *Managing Software Projects* ClearQuest: *Administrator's Guide*
UCM-PurifyPlus	Allows developers to invoke ClearCase from PurifyPlus.	PurifyPlus Help

Integration	Description	Where it is documented
UCM-RequisitePro	Allows RequisitePro administrators to create baselines of RequisitePro projects in UCM, and to create RequisitePro projects from baselines.	*RequisitePro User's Guide* RequisitePro Help *Using UCM with Rational Suite*
UCM-Rose	Stores Rose models in ClearCase.	Rose Help *Using UCM with Rational Suite*
UCM-Rose RealTime	Associates activities with revisions.	*Rose RealTime Toolset Guide* *Rose RealTime Guide to Team Development*
UCM-SoDA	Collects information from ClearCase and presents it in various report formats.	*Using Rational SoDA for Word* *Using Rational SoDA for Frame* SoDA Help
UCM-TestManager	Stores test assets in ClearCase.	*Rational TestManager User's Guide* TestManager Help *Using UCM with Rational Suite*
UCM-XDE	Stores XDE models in ClearCase	XDE Help
UCM-XDE Tester	Stores XDE Tester Datastores in ClearCase	XDE Tester Help

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 /opt/rational/clearcase on UNIX and C:\Program Files\Rational\ClearCase on Windows.
- *cquest-home-dir* represents the directory into which Rational ClearQuest has been installed. By default, this directory is /opt/rational/clearquest on UNIX and C:\Program Files\Rational\ClearQuest on Windows.
- **Bold** is used for names the user can enter; for example, command names and branch names.
- A sans-serif font is used for file names, directory names, and file extensions.
- **A sans-serif bold font** is used for GUI elements; for example, menu names and names of check boxes.
- *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 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, you can use "**...**" within a pathname as a wildcard, similar to "*" or "?". For more information, see the **wildcards_ccase** reference page.
- If a command or option name has a short form, a "medial dot" (·) character indicates the shortest legal abbreviation. For example:

  lsc·heckout

Online Documentation

The ClearCase Product Family (CPF) includes online documentation, as follows:

Help System: Use the **Help** menu, the **Help** button, or the F1 key. To display the contents of the online documentation set, do one of the following:

- On UNIX, type **cleartool man contents**
- On Windows, click **Start > Programs > Rational Software > Rational ClearCase > Help**
- On either platform, to display contents for Rational ClearCase MultiSite, type **multitool man contents**
- Use the **Help** button in a dialog box to display information about that dialog box or press F1.

Reference Pages: Use the **cleartool man** and **multitool man** commands. For more information, see the **man** reference page.

Command Syntax: Use the **–help** command option or the **cleartool help** command.

Tutorial: Provides a step-by-step tour of important features of the product. To start the tutorial, do one of the following:

- On UNIX, type **cleartool man tutorial**
- On Windows, click **Start > Programs > Rational Software > Rational ClearCase > ClearCase Tutorial**

PDF Manuals: Navigate to:

- On UNIX, *ccase-home-dir*/doc/books
- On Windows, *ccase-home-dir*\doc\books

Customer Support

If you have any problems with the software or documentation, please contact Rational Customer Support by telephone, fax, or electronic mail as described below. For information regarding support hours, languages spoken, or other support information, click the **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

Introduction to MultiSite

1

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 its own copy (replica) of the VOB. At any time, changes made in one replica can be sent in update packets to other replicas. The update process can be automated 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 not practical for all engineers to access the 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 VOB 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.

This chapter gives an overview of the major features in MultiSite. Chapter 2, *MultiSite Operation*, contains more details about how the features work.

Understanding the Architecture of MultiSite

The following sections describe the MultiSite architecture.

Replicated VOB Databases

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 information about 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 coordination issues that administrators must consider. (For more information, see Chapter 3, *Planning a MultiSite Implementation*.) Also, MultiSite features allow simultaneous development to occur at different replicas without conflicts. *Enabling Independent Development: Mastership* on page 5 describes how conflict avoidance works.

For more information, see *VOBs and VOB Replicas*, *VOB Objects and VOB Replica Objects* on page 8 and *ClearCase Commands Related to MultiSite* on page 61.

MultiSite Terminology

MultiSite documentation uses the following terms.

Term	**Definition**
Replica	A copy of a VOB. To refer to a VOB replica, use its replica name and VOB tag.
Family	All the replicas of a VOB. The family name of a VOB family is the VOB tag.
Site	The collection of clients and servers known to a registry host. Each site can contain at most one replica of a VOB.
Host	The LAN name or IP address of the network node that contains the database of the VOB replica.

VOBs and VOB Replicas

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.

Each replica is a copy of the VOB, including both file system data (data containers) and metadata (VOB database). At each replica, developers 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 identities and permissions differences or to propagate changes to identities and permissions from site to site. For more information, see *Identities and Permissions Strategy for VOB Replicas* on page 39.
- 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 them. 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. However, 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.

For more information, see *Information Propagated Among VOB Replicas* on page 7.

The biggest difference among replicas reflects the basic capability of MultiSite: enabling development work to proceed independently at different locations. For more information, see *Enabling Independent Development: Mastership* on page 5.

Synchronizing Replicas in a Family

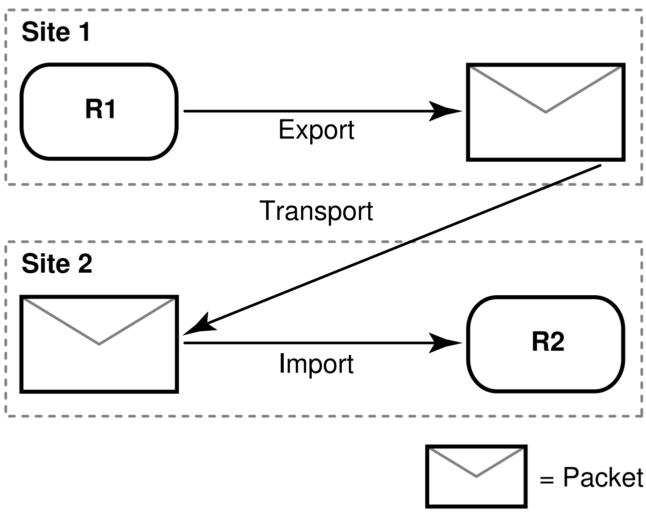
Because information in a replicated VOB is modified concurrently at different replicas, the contents of each replica in a family tend to diverge. In fact, the contents of a particular replica may never be identical to the contents of any other replica. To keep the replicas from diverging too much, each replica sends updates to one or more other replicas. Updating a VOB replica changes both its database and its storage pools to reflect the development activity that has taken place in one or more other replicas.

Information is exported from a replica in packets. A logical packet includes all the information required to create a new replica (replica-creation packet) or to update one or more existing 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 created with a **mkreplica** or **syncreplica** command invoked with the **–export** option and sent to a replica, it is processed at that replica by a **mkreplica** or **syncreplica** command invoked with the **–import** option. The changes that occurred originally at the sending replica (and perhaps at some other replicas, too) are added to the database and storage pools of the importing replica. 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 replica, unless the imports occur simultaneously.

Figure 1 illustrates the three phases of synchronization: export, transport, and import. At Site 1, a **syncreplica –export** command places records of operations from **R1** into a packet. The packet is sent to Site 2. At Site 2, a **syncreplica –import** command imports the contents of the packet into **R2**. Note that each synchronization is one-way. If two replicas update each other, two synchronizations are required.

Figure 1 Replica Synchronization



You can match the synchronization strategy for each family to its particular use patterns, your organization's needs, and the level of connectivity among the host machines. For one family, you can update replicas every hour, using a high-speed network; for another family, you can send updates only once or twice a month, using electronic mail or disk files as the delivery mechanism. For information about planning synchronization, see *MultiSite Use Model* on page 37. For information about creating and synchronizing replicas, see Chapter 7 and Chapter 8. *The Operation Log* on page 22 describes the mechanism that supports replication and synchronization.

Enabling Independent Development: Mastership

Because changes are made independently at multiple replicas, these changes may conflict. In a MultiSite environment, tracking changes and preventing data corruption are accomplished with an exclusive-right-to-modify scheme called *mastership*. Mastership determines when a user of a replica is allowed to modify data.

If the work done in different replicas were truly independent, the result would be chaos. Suppose version 17 of an element is created on the **main** branch in three replicas at the same time. Which is the real version 17, and what happens to the other versions?

Certain 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 10, *Managing Mastership*). In general, an object can be modified or deleted only at its master replica.

For example, each branch type has a master replica. By default, you can create or modify a branch only if your current replica masters the branch type. In this example, the command fails because the current replica does not master the **main** branch type:

cleartool checkout –c "add new feature" v3.0plan.doc
```
cleartool: Error: Unable to perform operation "checkout" in replica
"sanfran_hub" of VOB "/vobs/doc".
cleartool: Error: Master replica of branch "/main" is "boston_hub".
cleartool: Error: Unable to check out "v3.0plan.doc".
```

Replicas, VOBs, and most objects in a VOB have a master replica. For more information about how mastership prevents conflicting changes, see *Mastership* on page 9.

Some conflicts are unavoidable. For example, objects with the same name can be created at two or more VOB replicas during the same time period between synchronizations. You can minimize such conflicts by establishing naming conventions for objects, but if a conflict does occur, it is handled during import of an update packet. For more information, see *Conflict Resolution* on page 21.

Supporting Serial Development in VOB 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 so that development can proceed concurrently 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 of the master replica 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 11, *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 element.

For more information about enabling requests for branch mastership, see Chapter 11, *Implementing Requests for Mastership*. For more information about the use models for requesting mastership, see *Working On a Team* in *Developing Software*.

MultiSite Operation

2

This chapter provides more detail about the topics introduced in Chapter 1.

Information Propagated Among VOB Replicas

Some information is not replicated; in general, site-specific information is not replicated. Table 1 shows the information that is, and is not, propagated among replicas.

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

Data propagated	Data not propagated
Elements, branches, versions (including derived object versions).	Derived objects (DOs) that have not been checked in as versions. DOs tend to be large and short-lived; transmitting them among multiple replicas is likely to be less efficient than rebuilding them at each replica.
Most kinds of type objects.	Trigger type objects. Triggers are usually used to implement local policies, and trigger type definitions often include pathnames that do not exist at other sites.
Metadata annotations: version labels, attributes, hyperlinks (including merge arrows and hyperlinks to administrative VOBs).	Individual “attached” triggers.
UCM objects: activities, baselines, components, folders, projects, streams	
Permanent locks (those created with the **–obsolete** option).	Temporary locks (those created without the **–obsolete** option).

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

Data propagated	Data not propagated
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.
Event records.	
Mastership information. (See *Mastership* on page 9.)	Mastership request settings. See Chapter 11, *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.

VOB Objects and VOB Replica Objects

Each replica is a VOB, but the VOB object and VOB-replica object are different 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 (or 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
...
Vob family uuid: 87f6265b.72d911d4.a5cd.00:01:80:c0:4b:e7
Vob replica uuid: 87f6265f.72d911d4.a5cd.00:01:80:c0:4b:e7
```

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

cleartool describe vob:/vobs/dev replica:boston_hub@/vobs/dev

```
versioned object base "/vobs/doc"
  created 15-Aug-01.14:19:48 by susan.user
  "Main source VOB for development."
  master replica: boston_hub@/vobs/dev
  replica name: boston_hub
  VOB family feature level: 3
  VOB storage host:pathname "goldengate:/vobstg/dev.vbs"
  VOB storage global pathname "/net/goldengate/vobstg/dev.vbs"
...
replica "boston_hub"
  created 15-Aug-01.14:19:48 by susan.user
  replica type: unfiltered
  master replica: boston_hub@/vobs/dev
  request for mastership: enabled
...
```

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.

Mastership

The following sections describe how mastership applies to objects in databases.

Replica Mastership

When you create a new replica, its replica object (the 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, unless you transfer mastership to another replica.

To facilitate replica maintenance, we recommend that each replica be self-mastering, which means that it masters its own replica object. For more information, see *Transferring Mastership of a Replica Object* on page 132.

Note: To perform certain procedures on a replica object, you must make the replica self-mastering. This requirement is noted in the documentation for those procedures.

Branch Mastership

Branch mastership is the scheme that supports independent development work at different VOB replicas. Every branch type defined in a VOB (including the **main** branch type) has a master replica.

Mastership restrictions allow you to create a branch only if its creation will not conflict with an attempt to create a branch of that type at the replica that masters the type:

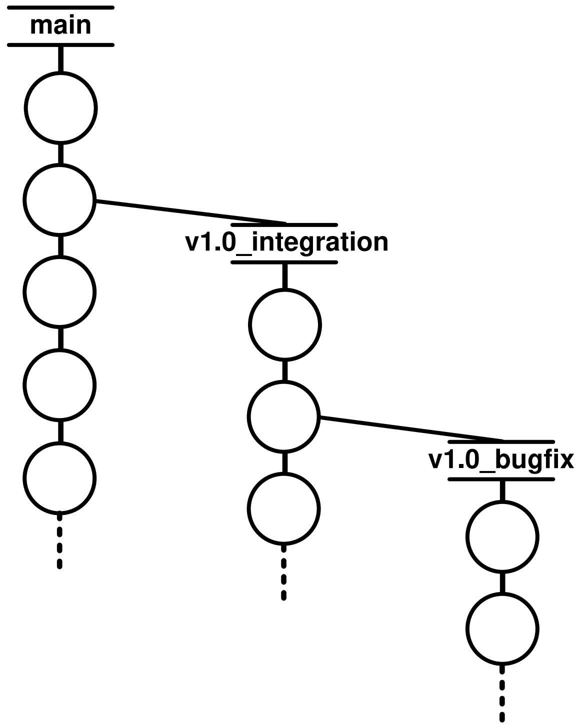
- Branches can be created only at the replica that masters the branch type, unless you are creating a new element. For more information, see *Creation of the main Branch of an Element* on page 12.
- By default, branches can be modified only at the replica that masters the branch type. Checking out a version is considered a branch modification. However, you can change mastership of an individual branch to another replica. You can also check out a version nonmastered, which means that the checkout succeeds even if your current replica does not master the branch. In order to check in the new version, you must request mastership of the branch and (if necessary) do a merge. For more information, see Chapter 11, *Implementing Requests for Mastership*.

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 strategy of using branches to isolate changes for particular development tasks. (For example, fixing a defect may require changes to five elements, in which each change is made on a branch of type **v1.0_bugfix**.) With Rational ClearCase MultiSite, work on various tasks can be done at different replicas, each using its own branch type. The work on different branches can be propagated among replicas, 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 replicas do not conflict.

Figure 2 shows a sample version tree. Each replica masters a branch type and developers using that replica can create versions only on the branch of that type. For example, the **boston_hub** replica masters the **main** branch type, the **sanfran_hub** replica masters the **v1.0_integration** branch type, and the **bangalore** replica masters the **v1.0_bugfix** branch type.

Figure 2 Branch Mastership



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

- By default, the replica at 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 create and 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 a branch of type **v1.0_integration** in 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 about supporting serial development with MultiSite, see *Supporting Serial Development in VOB Replicas* on page 5.

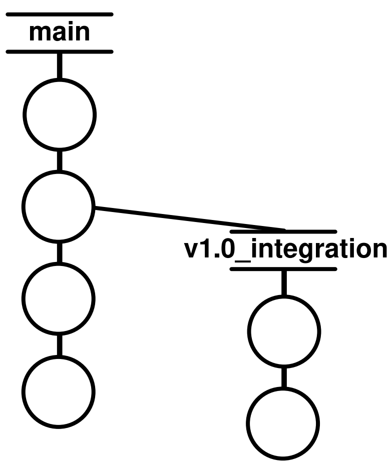
Creation of the main Branch of an Element

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 128.

Synchronizing Development on Different Branches

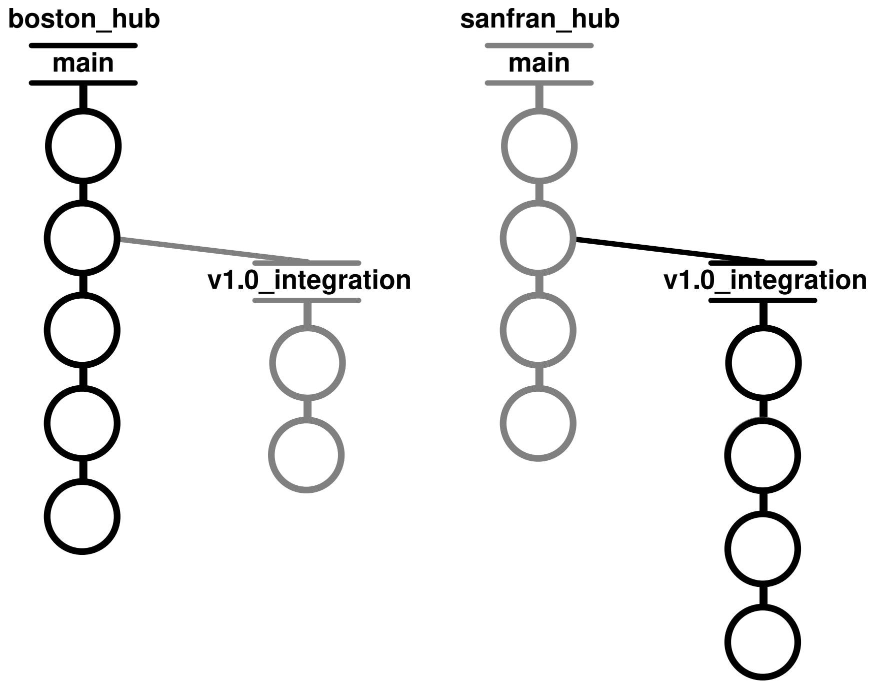
Development of an element with multiple branches can take place in different replicas concurrently, with occasional synchronizations. (The more frequently you update, the easier it is to track and reconcile the changes on different branches of elements. To reconcile changes, you use the ClearCase version-comparison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

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

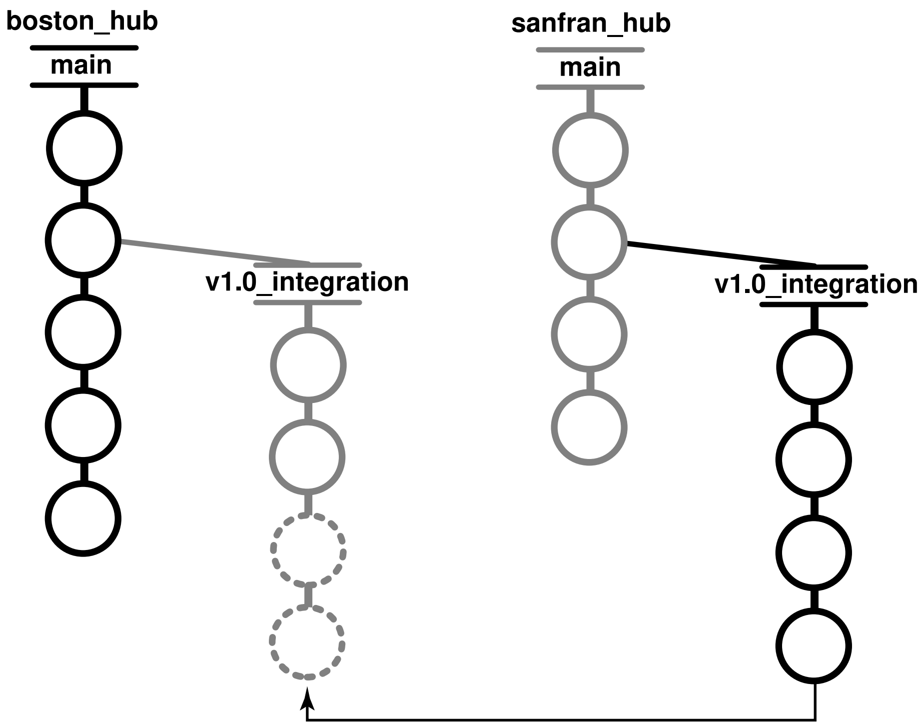


When the Boston site starts using MultiSite, the administrator creates a new replica for the San Francisco site. Because integration for Version 1.0 will be done at the San Francisco site, the **sanfran_hub** replica must master the **v1.0_integration** branch type. The administrator transfers mastership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

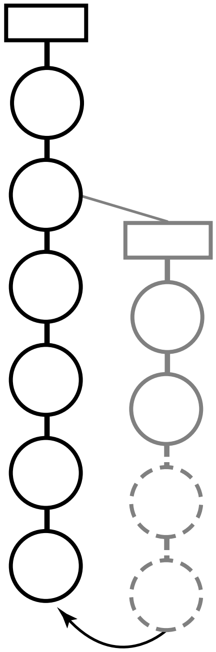
Developers in San Francisco can now create versions on existing branches of type **v1.0_integration** and can create new instances of that branch type. Work continues on the **main** branch in Boston:



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 which replica masters a branch and whether mastership is explicit or default.

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 136.

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 need to use the **TESTED_BY** attribute. Therefore, you can create two kinds of attribute type, hyperlink type, and label type objects: unshared and shared.

Unshared 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, attribute types, hyperlink types, and label types are created as unshared. An unshared type object can be converted to shared.

Shared Type Objects

Instances of a shared type object can be created in multiple replicas. To prevent cross-replica conflicts, the following restrictions apply:

- The instance restrictions (if any) on the type object must allow creation of the instance.
- For all objects except versions and branches, the current replica must master the object to which the instance is being attached.

- 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.

Note: If a hyperlink type is shared, you can create a hyperlink of that type between any two objects, at any replica. If the type is global, the restrictions on creation of the local copy apply.

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.

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.

Additional Restrictions for Shared Global Types

Additional mastership restrictions exist when you use an administrative VOB hierarchy and its global types. If a global type is shared, you can create instances of the type in a replica only if one of the following conditions exists:

- The replica contains a local copy of the type.
- The replica does not contain a local copy of the type, but the type is mastered by the administrative VOB replica at the current site. (If the type is not mastered by the administrative VOB replica, a local copy of the type cannot be created in the replica.)

These restrictions apply even if your current replica masters the object to which you are attaching the instance. These restrictions prevent conflicting, simultaneous creation of a given type with a given name at multiple sites. For more information about global types, see the *Administrator's Guide* for Rational ClearCase.

For more information about changing type mastership, see Chapter 10, *Managing Mastership*.

Creating an Instance of a Type

In summary, you can create an instance of a type in the following cases:

- The type is unshared, your current replica masters the type, and no instance restrictions exist.
- The type is shared, your current replica masters the target object, and no instance restrictions exist. If the type is global, a local copy of the type must exist in the replica, or the current administrative VOB replica must master the type.

Example

The administrator at the **boston_hub** replica 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 things 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 instance restrictions).

Mastership Restrictions for VOB Objects

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)

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
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
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

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
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
Object	Change event (**chevent**) Change mastership (**chmaster**) Change name (**rename**) Lock obsolete (**lock –obsolete**) Unlock (**unlock**)	Object
	Change protection (**protect**)	Object (if current replica preserves identities or permissions)
Project	Change (**chproject**) Remove (**rmproject**)	Project
Project VOB	Change list of promotion levels (**setplevel**)	**PromotionLevel** attribute type
Replica	Change host (**chreplica**) Change 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

Table 2 Mastership Restrictions for VOB Objects

Object	Action	Object your current replica must master
Type	Copy (**cptype**)	The replica containing the original type must master that type.
	Remove (**rmtype**) Replace (**mk***object***type** **–replace**)	Type
Version	Check in (**checkin**) Check out (**checkout**) Remove (**rmver**)	Branch With **checkout –unreserved –nmaster**, there are no mastership restrictions.
VOB	Change feature level (**chflevel**)	The replica to be changed must be self-mastering.
	Set up snapshots (**vob_snapshot_setup**)	The replica must be self-mastering.
VOB family	Change feature level (**chflevel**)	VOB object

Conflict Resolution

Mastership restrictions prevent most inconsistent changes in different replicas, but some are unavoidable. To avoid many naming conflicts, the administrators for a family must create and enforce naming rules for objects. A use model that is enforced consistently across sites reduces the potential for conflicts. For example, the administrators for a family follow these rules:

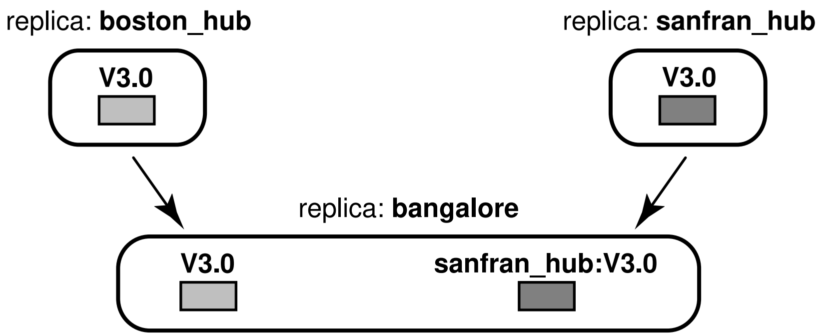
- All location-specific objects must include a location identifier.
- Objects that will be used at multiple replicas are all created at one replica.

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 196.

The Operation Log

This section describes the mechanism that supports 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 13, *Troubleshooting MultiSite Operations*.

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

- The identity of the replica in which the change originated.
- The change to the VOB database; for example, creation of a new element, checkin of a new version, attachment 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 oplog ID of the oplog entry.

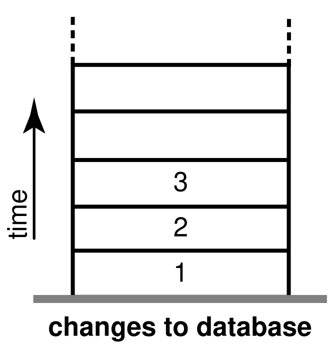
The exact kind and amount of information varies with the specific operation. For example, an oplog entry for the removal of a label has different, and less, information than an oplog entry for a **checkout** command.

Note: You can delete a replica's oplog entries after they have been used to update other replicas. For more information, see *Scrubbing Parameters for Replicas* on page 52.

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 a Database

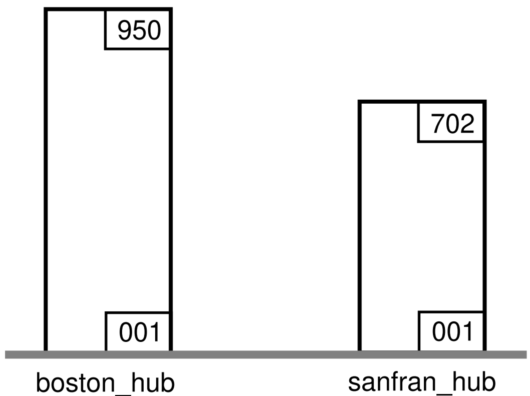


Within a replica family, changes are tracked for each replica. This is why an oplog entry includes the identity of the replica where the operation originated. Thus, the history of a replica family can be viewed as several stacks of oplog entries. Each stack is represented by a linear sequence of oplog IDs for the operations that originate in that replica.

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

- Operations with oplog IDs 1–950 have occurred at replica **boston_hub**.
- Operations 1–702 have occurred at replica **sanfran_hub**.

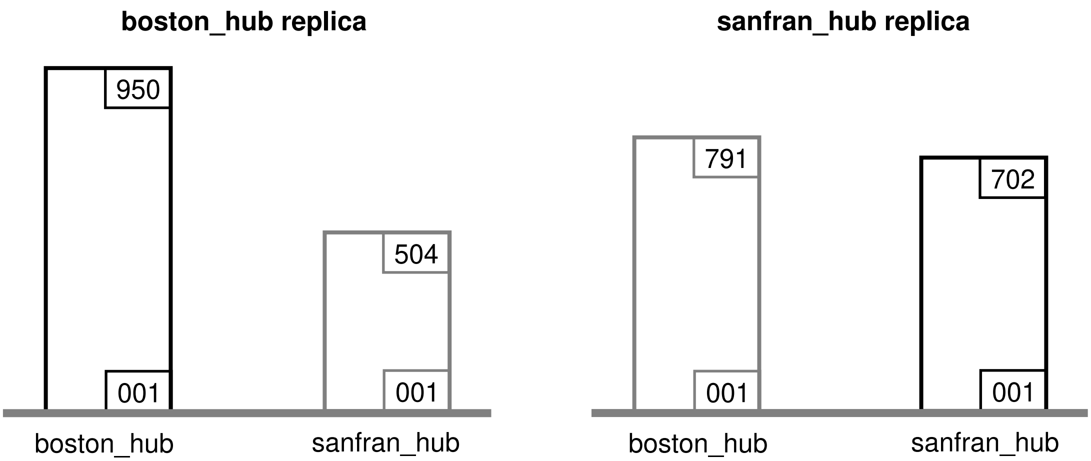
Figure 5 State of a Family



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 received update packets for only 791 **boston_hub** operations.

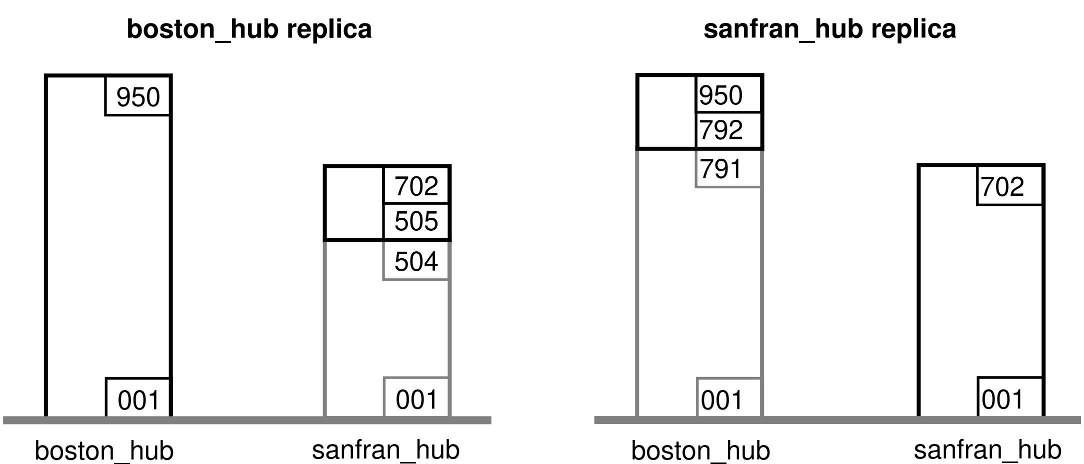
Figure 6 illustrates this scenario, in which each replica is out of date with respect to the operations originating at the other replica.

Figure 6 Out-of-Date Replicas



Picturing a replica family 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 operations originating at other replicas. (In certain error-recovery situations, you must reset a replica's data about its own operations. See Chapter 13, *Troubleshooting MultiSite Operations*.)

Figure 7 Updates Between Two Replicas



Note: By the time the packet is imported at **sanfran_hub**, additional changes may have been made at **boston_hub**. Those changes are not included in the update packet.

Oplog IDs and Epoch Numbers

An epoch number is the total number of operations that originated at a particular replica. In Figure 5, the epoch number for **boston_hub** is 950.

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

- **Changes made in the current replica**. The number of operations that originated at the current replica.
- **Changes at sibling replicas that have been imported to the current replica**. When **syncreplica** writes an operation from an update packet to the current replica, it increments the epoch number that records the number of operations originating at the sibling replica that have been imported at the current replica.
- **Estimates of the states of other replicas**. For each other replica, an estimate of its own changes and other replicas' changes. The current replica keeps track of the operations it has sent to other replicas, and assumes that these operations are imported successfully.

Table 3 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 epoch number is incremented.
- 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.
- 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 importing replica that the packet has been received and applied correctly. During normal 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 191.

Table 3 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

Table 3 Two-Row Epoch Number Matrix at Replica boston_hub

	Operations originated at boston_hub	**Operations originated at sanfran_hub**
boston_hub's estimate of **sanfran_hub**'s state	912	504

The contents of this matrix are reported by the **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 the **boston_hub** replica, the number of local operations is 950 (number in upper left corner of matrix), and the estimate is that the **sanfran_hub** replica has imported all operations through oplog ID 912 (number in lower left corner).

2 The update packet that the **boston_hub** replica sends to the **sanfran_hub** replica 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 a family includes more than two replicas, synchronization can occur indirectly. A replica can include nonlocal changes in update packets. For example, if the **boston_hub** replica exchanges updates with the **sanfran_hub** and **bangalore** replicas, 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 its family, its rows for those replicas may contain zeros. This is expected behavior.

Table 4 shows replica **boston_hub**'s epoch number matrix.

Table 4 Three-Row Epoch Number Matrix at Replica boston_hub

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

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 the Boston site uses this matrix to export an update for the **bangalore** replica:

1. At the **boston_hub** replica, there are 950 local operations (number in upper left corner of matrix), and the estimate is that the **bangalore** replica has imported all operations through oplog ID 709 (lower left corner).
2. For operations that originated at the **sanfran_hub** replica, **boston_hub** has imported all operations up to oplog ID 504 and estimates that **bangalore** has imported all operations through oplog ID 221.

3 The update packet that **boston_hub** sends to **bangalore** includes **boston_hub** operations 710-950 and **sanfran_hub** operations 222-504. The output of an **lsepoch** command at the **boston_hub** replica 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)
```

Planning a MultiSite Implementation

3

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 and implement your design decisions in a set of test replicas before changing your development environment.

MultiSite Installation

For MultiSite installation instructions, see the *Installation Guide*.

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 host where replicas will reside or where the shipping server will be used 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 5 describes the amount of available disk space needed on the disk partition where the storage bay is located.

Table 5 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 the 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, size of largest packet to be stored in the bay.

There is no formula for determining how large your update packets will be. The general rule is that synchronizing more frequently usually results in smaller packets. Yet even if you synchronize every hour, a large amount of development activity or release activity can occur in an hour and a large packet will 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, see the **syncreplica** and **sync_export_list** reference pages.

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

MultiSite Licensing

A MultiSite license is required for any access to an object in a replica—by a MultiSite command or GUI, by a ClearCase command or GUI, or by a standard operating system command. You can calculate the number of MultiSite licenses you need by determining how many developers will access replicated VOBs. If all developers will access these VOBs, you need the same number of MultiSite licenses as ClearCase licenses. If some developers will not 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 will be replicated to site B and one will not be replicated. Five developers at site A will access only the unreplicated VOB, and the other 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 one 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 the company purchased three ClearCase licenses for site B, it would also purchase three MultiSite licenses for site B.

For more information about acquiring and setting up licenses, see the *Installation Guide* for the ClearCase Product Family.

Shipping Server Use with ClearCase and ClearQuest

If you use both Rational ClearCase MultiSite and Rational ClearQuest MultiSite, you use the same shipping server for both products. The shipping server is installed when you install ClearCase MultiSite.

Note: If you use ClearQuest MultiSite and ClearCase, or ClearQuest MultiSite alone, you must install the shipping server. For more information, see the installation information for Rational ClearQuest.

The following restrictions apply when you use both ClearCase MultiSite and ClearQuest MultiSite:

- You must use different storage classes for VOB replica packets and ClearQuest database replica packets. You can create multiple storage classes and use the **–sclass** option to specify a particular class. If you do not use the **–sclass** option, the default class is used:
  - For ClearCase MultiSite, the default storage class is **–default**. It is created when you install ClearCase MultiSite.
  - For ClearQuest MultiSite, the default storage class for **multiutil** commands that use the **–sclass** option is **cq_default**. The **shipping_server** and **mkorder** commands use **–default** as the default class.

    The **cq_default** class is not created during installation. If you plan to use this class, you must create the class and its shipping and return bays. For more information, see the **shipping.conf** (UNIX) and **MultiSite Control Panel** (Windows) reference pages.

If you do not create the **cq_default** storage class, you must create another class for use with ClearQuest MultiSite, and use the **–sclass** option in **multiutil** commands to specify that storage class. If the **cq_default** storage class does not exist and you do not specify the **–sclass** option in a **multiutil** command, the packet is placed in the storage bay associated with the **–default** class, which can cause problems at the importing site.

- You must use different bays for ClearQuest MultiSite storage classes and ClearCase MultiSite storage classes.
- If you uninstall one product, the other one may stop working. You must uninstall both products and then reinstall the one you want to continue using.

We recommend that you follow these guidelines when you use both ClearCase MultiSite and ClearQuest MultiSite:

- When you export a packet for a ClearQuest replica, use the **–sclass** option and specify a storage class.
- Enable e-mail notification for shipping server operations and specify an address to use only for messages originating from ClearQuest MultiSite operations. For more information, see the **control_panel** reference page in the *Administrator's Guide* for Rational ClearQuest MultiSite.

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.

Another approach is to use a single release integration branch, multiple site integration branches, and multiple developer branches. 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 replicas:

Model 1. Create a schedule that determines when each replica 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 11, *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/0 -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 128.

Use of Attributes, Labels, and Hyperlinks

Mastership restrictions affect the way you use ClearCase attributes, labels, and 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, with additional restrictions if you are using global types. For more information, see *Type Object Mastership* on page 16.

Use of Triggers

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.

Use of Multiple Replicas of the Same VOB at a Site

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. Do not assign public VOB tags in the same ClearCase registry region to multiple replicas of the same VOB.

For information about how VOBs and VOB replicas are listed in the ClearCase storage registry, see *VOB Objects and VOB Replica Objects* on page 8. For information about using multiple replicas at one site, see Chapter 12, *Using MultiSite for VOB Backup and Interoperability*.

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

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

Additional mastership restrictions exist when you use an administrative VOB hierarchy and its global types. If a global type is shared, you can create instances of the type in a replica only if one of the following conditions exists:

- The replica contains a local copy of the type.
- The replica does not contain a local copy of the type, but the type is mastered by the administrative VOB replica at the current site. (If the type is not mastered by the administrative VOB replica, a local copy of the type cannot be created in the replica.)

These restrictions apply even if your current replica masters the object to which you are attaching the instance. These restrictions prevent conflicting, simultaneous creation of a given type with a given name at multiple sites.

Use of UCM

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.

The following restrictions apply to use of UCM and MultiSite:

- You cannot request mastership of branches or branch types that are associated with streams.
- If you replicate a UCM component, you must replicate its associated UCM project VOB (PVOB).
- You must synchronize a UCM component and its associated PVOB at the same time.
- UCM projects enabled for ClearQuest can be replicated and synchronized. In addition to using ClearCase MultiSite to replicate and synchronize UCM project and component VOBs, you can use ClearQuest MultiSite to replicate and synchronize associated ClearQuest user databases. You must synchronize a UCM PVOB and its associated ClearQuest user database at the same time.

MultiSite Use Model

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

Type of Administration

You must 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 family, all its replicas 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 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 ensure that schedules do not conflict.

  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, sites with major development efforts have local MultiSite administrators, and responsibility for administering smaller sites is distributed among the MultiSite administrators.

MultiSite, Time, and Time Zones

In 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, India, checks in a version at 14:33 Bangalore time, the creation time is stored as 09:03UTC. When a developer in San Francisco looks at the version, the 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 *Synchronization Schedule* on page 50.

Time Rules in Config Specs

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 replica. 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 replica created version 18 on July 10 at 6:00 P.M. your time, but this change has not been propagated to your replica. After the update packet that contains the change is imported at your replica, your time rule selects version 18.

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 objects. After you create the replicas in the family, you can change mastership of objects. For more information, see *Enabling Independent Development: Mastership* on page 5 and *Changing Mastership of VOB Objects* on page 130.

Identities and Permissions Strategy for VOB Replicas

When you import a replica-creation packet, you must specify a preservation mode for the new replica. A replica can preserve identities and permissions, preserve permissions only, or preserve neither identities nor permissions. In most cases, your replicas must be permissions preserving or nonpreserving.

The following sections describe the three modes.

Identities- and Permissions-Preserving Replicas

Identities- and permissions-preserving replicas maintain the same user and group identities and permissions on elements, and changes to identities or permissions are synchronized among them. The owner and group of the original VOB are not preserved; the user who enters the **mkreplica –import** command becomes the owner of the new VOB. That user's group is the primary group of the VOB, and the user's group list becomes the VOB's group list. The user must be a member of all the groups that are used for elements in the replica.

To create a replica that preserves identities and permissions, you should run **mkreplica –export** at an identities- and permissions-preserving replica.

Note: You may need to run **cleartool protectvob** on the new VOB replica to ensure that the owner, group, and group list of the new VOB match the values in the other identities- and permissions-preserving replicas in the VOB family. Make sure that you update the VOB owner's group list. When you run **mkreplica –import**, you may want to run the command as the VOB owner of the other identities- and permissions-preserving replicas in the family. If the VOB owner account has the same group list at the exporting and importing sites, you do not need to run **protectvob**.

Permissions-Preserving Replicas

Permissions-preserving replicas maintain the same permissions on elements, and changes to permissions are synchronized among the other replicas in the family that preserve permissions, including those that preserve both identities and permissions.

Read, write, and execute permissions for user, group, and other are preserved. The set-UID bit, set-GID bit, and sticky bit are not preserved.

Caution: If you need to restrict read or execute permissions to certain subgroups, we recommend that you do not use permissions-preserving replicas. It is possible for a malicious user at one site to change the permissions on an element in order to grant read access to a user at another site who is not the element owner or in the element's group. If you choose to use permissions-preserving replicas, you may want to define a trigger that informs you when a **cleartool protect** command is run. Also, when you run **mkreplica –import** and create a permissions-preserving replica, make sure that your primary group is appropriate.

In order for you to change a replica to be permissions preserving or to create a new permissions-preserving replica, the VOB family feature level of the replica must be 4. Also, the **cleartool protect** command fails if you use the **–chmod** option, specify an object in a permissions-preserving replica, and your client host is running a version of ClearCase associated with feature level 3 or lower.

Permissions-preserving replicas ignore changes to identities made at other replicas and maintain their own identities information for elements. For permissions-preserving replicas:

- The user who enters the **mkreplica –import** command becomes the owner of the new VOB and of all elements in it. When you import an update packet containing oplogs for new elements, the VOB owner becomes the owner of the new elements.
- The primary group of the user who enters the **mkreplica –import** command becomes the VOB's primary group and the group for all elements. When you import an update packet containing oplogs for new elements, the VOB's primary group is the group for the new elements.
- Changes to identities are not propagated to other replicas. Identities changes made at replicas that preserve identities are ignored at permissions-preserving replicas.

To create a replica that preserves permissions, you should run **mkreplica –export** at an identities- and permissions-preserving replica or a permissions-preserving replica.

Nonpreserving Replicas

Nonpreserving replicas ignore identities and permissions changes made at other replicas and maintain their own identities and permissions information for elements. For nonpreserving replicas:

- The user who enters the **mkreplica –import** command becomes the owner of the new VOB and of all elements in it. When you import an update packet containing oplogs for new elements, the VOB owner becomes the owner of the new elements.
- The primary group of the user who enters the **mkreplica –import** command becomes the VOB's primary group and the group for all elements. When you import an update packet containing oplogs for new elements, the VOB's primary group is the group for the new elements.
- The initial permissions of the elements are the same as their values in the replica at which the **mkreplica –export** or **syncreplica –export** command is entered.
- Changes to identities and permissions are not propagated to other replicas. Changes made at replicas that preserve identities and permissions or permissions only are ignored at nonpreserving replicas.

Synchronization of Identities and Permissions Information

When an update packet is imported at a permissions-preserving replica, identities information is ignored. When an update packet is imported at a nonpreserving replica, identities and permissions information is ignored. At both kinds of replicas, the information remains in the oplog entries so that it can be transmitted to replicas that preserve identities and permissions or permissions only. For example:

1. A new element is created at replica **A**, which preserves identities and permissions.
2. Replica **A** sends an update packet to replica **B**, which is nonpreserving.
3. The new element is created at replica **B**. Its owner is the VOB owner of replica **B** and its group is the VOB's primary group.
4. Replica **B** sends an update packet to replica **C**, which preserves identities and permissions.
5. The new element is created at replica **C**. Its owner is the original creator and its group is the original creator's group.

Elements created at a nonpreserving or permissions-preserving replica always get the importing VOB's owner and group when they are imported, regardless of whether the importing replica preserves identities.

Requirements for Replicas That Preserve Identities and Permissions

The sites of replicas that preserve both identities and permissions 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 (UIDs and GIDs) are not consistent across domains.

Therefore, the entire set of replicas cannot preserve identities 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 preserve identities in 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 identities-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 registered on two different ClearCase registry hosts). **boston** and **boston_backup** are created as identities-preserving replicas.

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

Gathering Identities Information

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

1 At the exporting site, gather the current VOB owner 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. In this example, the owner is `ccadm` and the group is `user`:

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, as the VOB owner, issue the **id** command. For example:

su ccadm

`Password:` **xxxxxx**

id

```
uid=1083(ccadm) gid=20(user)
```

On Windows, as the VOB owner, issue the *ccase-home-dir***etc\utils\creds** command. For example:

`C:\>` **"Program Files\Rational\ClearCase\etc\utils\creds"**

2 At each importing site, ensure that the importing user's 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. (These kinds of import failures are described in *Preservation Mode* on page 194.)

   If the names are the same and the numbers are different, you must create nonpreserving or permissions-preserving replicas.

Running protectvob on Identities-Preserving Replicas

If you run **protectvob** on a VOB replica that preserves identities, you must follow these steps to prevent metadata divergence or synchronization problems among replicas in the VOB family:

1. Stop synchronization among identities-preserving replicas in the family. Make sure that all update packets have been imported.
2. Run the **protectvob** command on all identities-preserving VOB replicas in the family. You must use the same options and arguments in each command.
3. Restart synchronization.

If you do not change the owner or group at all identities-preserving VOB replicas at the same time, metadata divergence can occur for the owner or group of new elements created at nonpreserving replicas. When the oplog entry for the new element is imported at an identities-preserving replica, the element's owner or group is the owner or group of the VOB at the time the entry is imported. If a change to the VOB owner or group has been made at other identities-preserving replicas, the element's owner or group will be different at the different replicas.

If you do not add a group to all identities-preserving VOB replicas at the same time, synchronization failures can occur for elements with the new group. The failures occur during import at the replicas where the group was not added.

When you remove a group from the group list of a VOB replica, the group cannot be used for new elements created in the VOB, but existing elements with that group are not changed. (To find these elements, use the **find** command with the **–group** option.) If you do not remove the group from all identities-preserving replicas at the same time, synchronization import failures can occur for new elements created with the deleted group. (You can fix the synchronization failure by running **protectvob –add_group** on the importing VOB replica.) An alternative to using **protectvob –delete_group** is to leave the group in the VOB's group list and create a trigger that checks the primary group of the user and prevents creation of the element if the user's primary group is one of the obsolete groups.

Synchronization Transport Method

There are several methods for transporting update and replica-creation packets. The method you choose depends on how your sites are connected, how quickly you must transfer packets, and how important security is. For more information, see Chapter 5, *Choosing a Transport Method*.

Synchronization Pattern

The synchronization pattern for a family defines which replicas exchange update packets and the direction of exchange. Figure 1 on page 4 shows a simple synchronization pattern, 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:

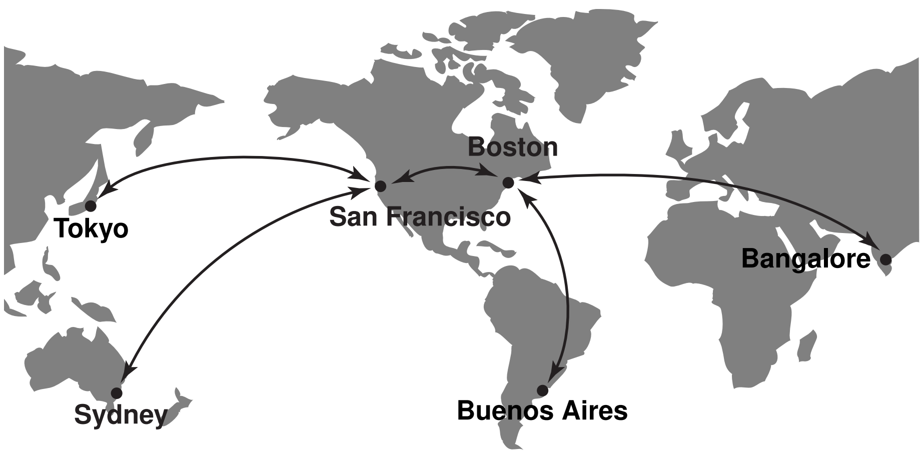
Update 1: Replica **1** sends changes to Replica **2**
Update 2: Replica **2** sends changes to Replica **3**

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 you flexibility in devising update strategies and patterns. For efficiency, a single update can be targeted at multiple replicas, for example, all other replicas in a family.

In general, you can implement any update topology, as dictated by organizational structures, communications/transportation costs, and so on. Figure 8 shows a simple peer-to-peer synchronization pattern, and Figure 9 shows a double-hub hierarchical pattern.

Figure 8 Peer-to-Peer Synchronization Pattern

Figure 9 Hierarchical Synchronization Pattern



Your choice of pattern depends on the following factors:

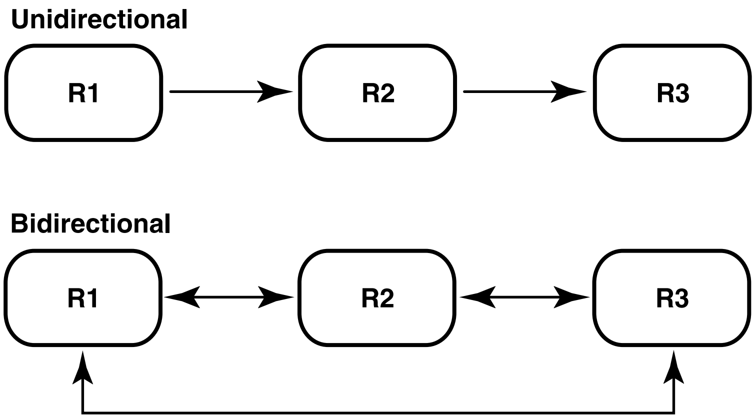
- Bandwidth between sites
- Network topology
- Latency of changes: how quickly changes made at one replica need to be received at another replica in the family
- Failure tolerance

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 synchronization. Unidirectional synchronization is suitable in situations like these:

- You use a replica as a backup.
- Your company supplies information to another site (or company) for read-only use.
- A high-security development project uses the same data 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.

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; you can do this by creating triggers or locking the VOB.

One-to-One and Ring Synchronization

Figure 11 One-to-One Synchronization Pattern

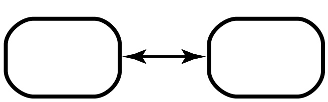
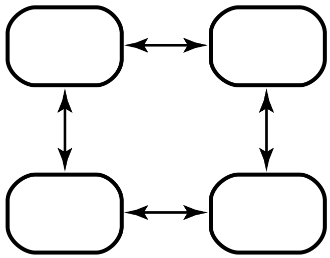


Figure 12 Ring Synchronization Pattern



The one-to-one and ring (or round-robin) patterns in Figure 11 and Figure 12 are simple patterns that are most suitable for small numbers of replicas. As the number of replicas increases, so does the amount of time for changes originating 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

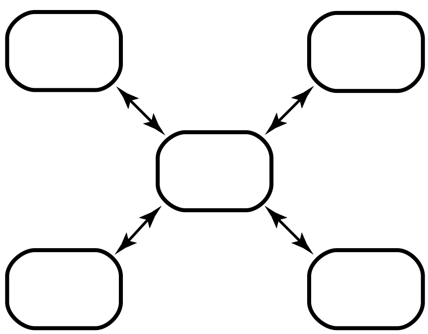


Figure 14 Multiple-Hub Synchronization Pattern

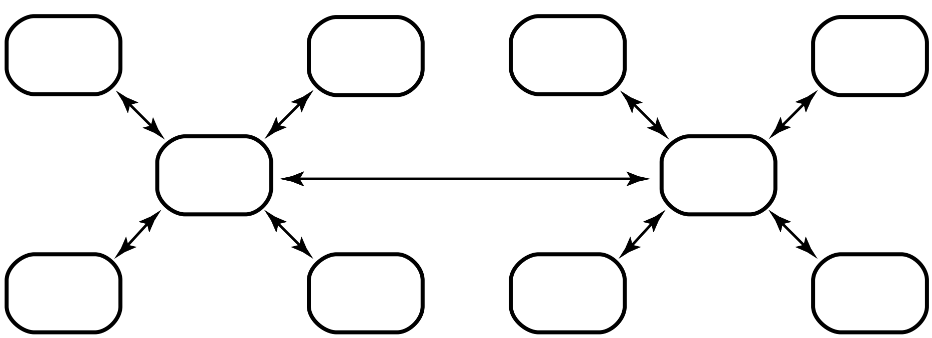
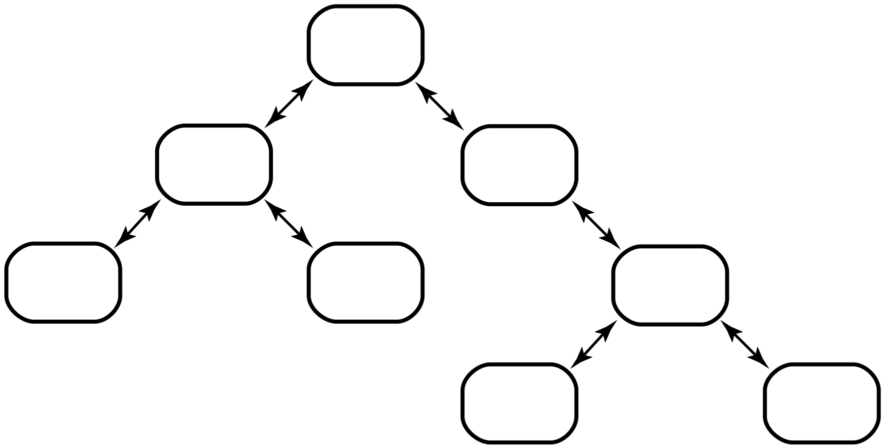


Figure 15 Tree Synchronization Pattern



In the hub patterns (Figure 13 and Figure 14), the hub replicas exchange packets with all spoke replicas. In the tree pattern (Figure 15), the root replicas exchange packets with branch replicas.

Advantages:

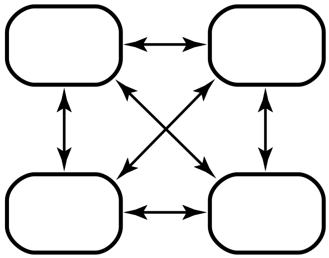
- More efficient for the spoke and branch replicas, which send to and receive from only one other replica.

Disadvantages:

- If the hub or root site goes down, all spoke/branch sites must reconfigure their pattern to continue communication.
- 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 in VOB replica families, you can run **chepoch –actual** regularly among the spoke or branch replicas. For more information, see the **chepoch** and **sync_export_list** reference pages.

Many-to-Many Synchronization

Figure 16 Many-to-Many Synchronization Pattern



In the many-to-many synchronization pattern (Figure 16), each replica exchanges packets with all other replicas

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 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, you lose less work if a replica is deleted accidentally and you must restore it from backup. Also, merging is simpler because fewer changes have been made.

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

- Time zone differences

Take different time zones into account when you send an update or set up automated updates. Figure 17 illustrates synchronization among replicas in multiple time zones.

- Use of administrative VOBs

  Because local type objects in VOBs are linked to global type objects in the administrative VOB, we recommend that you synchronize VOBs and their administrative VOB at the same time.

  For example, at the Boston site, the 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.

  If you do not synchronize VOBs and their administrative VOBs at the same time, users may have trouble accessing type objects.

- Use of ClearCase UCM

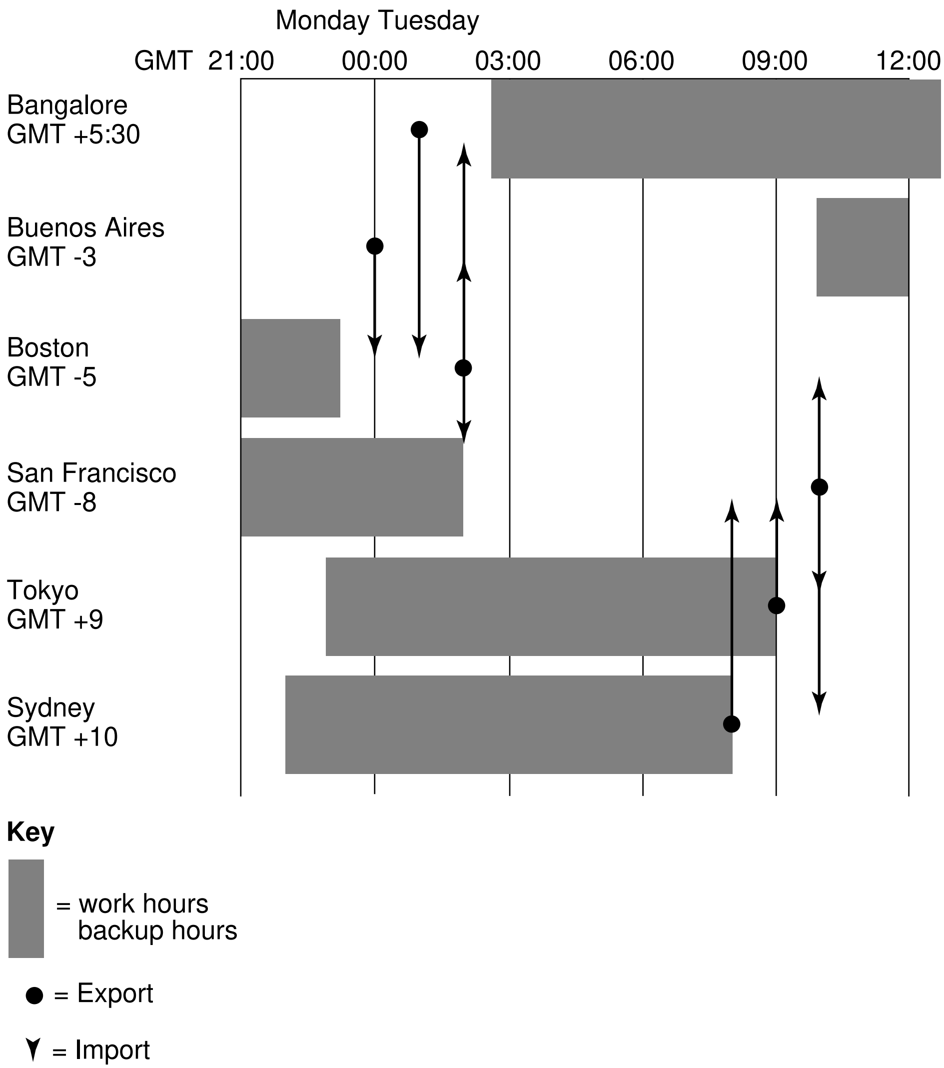
  We recommend that you synchronize a component VOB and its PVOB at the same time. If you do not, users may have trouble accessing baselines and activities and the versions associated with those objects.

For example, the administrators for the family in Figure 9 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 sends an update packet to the hub replica, and then the hub replica sends 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 backups.
- All replica hosts use receipt handlers to import packets as soon as they are received.

Figure 17 shows the synchronization timeline for the hub-spoke updates (but not the hourly 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 17 A Synchronization Schedule



Use of MultiSite for VOB Backups

You can use MultiSite as part of your VOB backup strategy. For more information, see Chapter 12, *Using MultiSite for VOB Backup and Interoperability*.

Scrubbing Parameters for Replicas

When a command makes a change to a replica, an entry is recorded in the replica's operation log. For more information about this mechanism, see *The Operation Log* on page 22. Also, when you export an update packet, an export_sync record is created for each target replica. These records 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 about VOB scrubbing, see the **vob_scrubber** reference page.

Oplog Scrubbing

Oplog entries must be kept for a significant period of time. They are required when the replica generates update packets. Oplog entries also 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 199.) 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 family and verify that packets are being exported and imported successfully.

When it is safe to delete oplog entries for a replica:

1. Coordinate with other administrators to decide how long you must keep oplog entries.

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

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

     Frequency of synchronization refers to both how often updates are exported and how often they are imported at other replicas. Also, consider setting up a verification scheme to 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 replica 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. Create a scrubber parameter file specific to your VOB by copying *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.
3. Make this new file writable.
4. 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 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 backups. (See *Recovering from Lost Packets* on page 190.)

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.

Handling Pathnames That Contain Spaces

On Windows, if the pathname of a receipt handler or a shipping order contains spaces, DOS "short name" resolution must be enabled for the file system on which the receipt handler or shipping order is located. This property is enabled by default. If this property is not enabled, the shipping server cannot invoke the receipt handler or process the shipping order.

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 replica creation and synchronization

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

- Monitor ClearCase and system log files

  Error and status messages are written to the **shipping_server_log** file on UNIX and the Event Viewer on Windows. For more information about error logs, see *Troubleshooting Tips* on page 173.

- 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 *Installation Guide* for the ClearCase Product Family.

- Coordinate issues with all other MultiSite administrators

  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 and record information about a family. Table 6 shows an example of information that may be useful. You may also want to draw a picture of the family's synchronization pattern.

Table 6 Family Information

Replica name	Replica host	Administrator	E-mail, phone number	Location	Time zone offset
sanfran_hub	goldengate	John Cole	jcole, x1462	San Francisco, CA, USA	GMT-8
boston_hub	minuteman	Susan Goechs	susan, x3742	Boston, MA, USA	GMT-5
tokyo	shinjuku	Masako Ito	masako, x7761	Tokyo, Japan	GMT+9
sydney	taronga	Bruce Fife	bfife, x5080	Sydney, Australia	GMT+10
bangalore	ramohalli	Sonia Kumar	kumar, x2347	Bangalore, India	GMT+5:30
buenosaires	mardelplata	Juan Fangio	fangio, x4300	Buenos Aires, Argentina	GMT-3

- Ensure that replicas receive any necessary special handling

  Restoring a replica from backup is a significant event. Failure to follow the procedure described in the section *Restoring a Replica from Backup* on page 199 leads to irreparable inconsistencies among the replicas in a family.

  There are no special requirements for backing up a replica. Use the backup instructions in the *Administrator's Guide* for Rational ClearCase. Other ClearCase administrative procedures require special steps for replicas. The procedures in the *Administrator's Guide* describe these steps.

MultiSite Command Set

4

This chapter summarizes the commands for Rational ClearCase MultiSite and Rational ClearCase that display MultiSite information. Reference pages for the MultiSite commands are available in Chapter 14, *MultiSite Reference Pages*, and are also available online:

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

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.

multitool Use

The **multitool** command is used to perform operations on VOB replicas. The command has the following features:

- It has a set of subcommands that perform product functions, such as replica creation, synchronization, and management; mastership changes of objects; and failure recovery.
- Some subcommands and command options can be truncated, as indicated in the reference pages.

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

  multitool lspacket

  Also in interactive mode:

  multitool
  `multitool>` **lspacket**
  `multitool>` **quit**

- Commands and options are case sensitive and must be typed in lowercase.
- The **help** command displays syntax summaries.

  multitool help chreplica

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

- The **man** command displays reference pages:

  multitool man chreplica
  ...on Windows, the reference page is displayed in a Web browser

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

  APPLICABILITY
  ...
  ```

- With the **contents** argument, the **multitool man** command displays the HTML table of contents for the MultiSite Help system:

  multitool man contents

Descriptions of Subcommands

The following sections describe the different kinds of **multitool** subcommands.

Replica Creation, Synchronization, and Management Commands

The commands in Table 7 create new replicas, change replica characteristics, and synchronize replicas.

Table 7 Replica Creation, Synchronization, and Management Commands

Command	Description
chreplica	Changes the properties of a replica
lspacket	Lists one or more packet files created by **mkreplica** or **syncreplica**
lsreplica	Lists one or more replicas
mkreplica	Creates a new replica
rmreplica	Removes a replica
syncreplica	Synchronizes a replica with one or more replicas in its family

Object Mastership Commands

To avoid introducing conflicting changes at different replicas, certain objects are assigned a master replica (master). The initial master of an object is the replica where the object is created. For more information about mastership, see *Enabling Independent Development: Mastership* on page 5. Table 8 lists the commands you can use to manage mastership.

Table 8 Object Mastership Commands

Command	Description
chmaster	Transfers mastership of an object
lsmaster	Lists objects mastered by a replica
reqmaster	Requests mastership or set access controls for mastership requests

Failure Recovery Commands

Each replica 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 has 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 *The Operation Log* on page 22.

Use the failure-recovery commands in Table 9 when this assumption of successful delivery does not hold true.

Table 9 Failure-Recovery Commands

Command	Description
chepoch	Changes a replica's epoch number matrix
lsepoch	Lists a replica's epoch number matrix
recoverpacket	Resets a replica's epoch number matrix so lost packets are resent (required when a packet is lost or unusable)
restorereplica	Restores a 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.

multitool Utility Commands

The commands in Table 10 are also **cleartool** commands and are documented only in the *Command Reference*, except for **apropos**, which is also documented in this manual.

Table 10 multitool Utility Commands

Command	Description
apropos	(UNIX) Displays **multitool** command information
cd	Changes current working directory
describe	Describes a replica's VOB database object
help	Displays **multitool** command syntax
man	Displays a MultiSite reference page
pwd	Prints working directory
quit	Ends interactive **multitool** session
rename	Renames a replica
shell	Creates subprocess to run shell or program

Additional MultiSite Commands

The MultiSite commands that are not **multitool** subcommands are listed in Table 11.

Table 11 Additional MultiSite Commands

Command	Location under *ccase-home-dir*	Description
epoch_watchdog	config/scheduler/tasks	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	Replica-update script using store-and-forward; for use in **schedule** commands
sync_receive	config/scheduler/tasks	Replica-update script using store-and-forward; for use in **schedule** commands and as the receipt handler

ClearCase Commands Related to MultiSite

The ClearCase commands in Table 12 manage or display MultiSite information.

Table 12 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 in all replicas of a VOB (Default: lists your current replica's checkouts)
mkattype –shared **mkhltype –shared** **mklbtype –shared**	Creates a shared type object (For more information, see *Shared Type Objects* on page 16)

Table 12 ClearCase Commands Related to MultiSite

Command	Description
mkelem –master **mkdir –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, **mkelem** or **mkdir** 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
: You can change a directory only 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*object***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.

View Contexts and VOB Mounts

Most MultiSite commands do not require a view context or mounting of the VOB replicas being processed. However, 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.

The following **multitool** commands require a view context:

- **describe**
- **chmaster** (for file system objects)
- **lsmaster**
- **reqmaster**

Specifying VOBs in Commands

multitool commands use the *vob-selector* argument to specify a VOB family. *vob-selector* is also used as a suffix in object selectors. In most cases, if you do not specify a *vob-selector*, the command uses the VOB containing the current working directory.

MultiSite Configuration

Choosing a Transport Method 5

This chapter describes the methods for transporting packets between replicas. The method you choose depends on connectivity between replicas. If your replicas do not have IP connectivity, you must use a file-based method. If your replicas have connectivity, you can use the store-and-forward facility of Rational ClearCase MultiSite.

Table 13 lists the recommended methods for various situations.

Table 13 Choosing a Packet Transport Method

Your situation	Recommended methods
Sites are connected with high-speed lines	Store-and-forward
One or more sites have firewalls	File-based methods (e-mail, **ftp**, physical media), store-and-forward
Must transfer packets quickly	File-based methods (e-mail, **ftp**), store-and-forward
No electronic connection between sites	File-based methods (physical media)

File-Based Methods

These transport methods include e-mail, **ftp**, and physical media (like CDs, magnetic tapes, and diskettes).

Using Electronic Mail

You can use an existing electronic mail mechanism as the transport method for packets. On the sending end, compress and encode the 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 specify the maximum size for a packet 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 intercepted or lost without notification.
- Less efficient than **ftp** or store-and-forward.

Notes:

- You can write scripts to automate e-mail transport. The sending script creates the 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 location or replica.

  At the target location, the account that receives the packets redirects or pipes them to a process that reassembles, decodes, and uncompresses them and 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 between replicas. 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 that is accessible by MultiSite administrators at other sites. Scripts at receiving sites poll the drop site, looking for any new files. When new files arrive, the scripts retrieve them using **ftp**, decompress them, and process them.

Advantages:

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

Disadvantages:

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

Using Physical Media

You can create packets as files, write them to a CD, magnetic tape, or diskette, and then send the media to another site. The **mkreplica** and **syncreplica** commands include the **–out** option, which places packets in physical files.

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 ensure that 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.
```

Store-and-Forward

The MultiSite store-and-forward facility (the shipping server) is a file-transfer service that automates the transport phase of replica creation and synchronization. It 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. This is how the store-and-forward process works:

1. During the export phase, a packet file and a shipping order file are created. The shipping order file contains delivery instructions for the packet.
2. The packet and shipping order are stored in one of the storage bay directories on the VOB replica host.

If the packet is associated with a storage class, the packet is stored in the storage bay specified by the storage class. You can define storage classes in the shipping.conf file on UNIX and the MultiSite Control Panel on Windows.

3. The shipping server uses the instructions in the shipping order to transfer the packet file from the storage bay at the local site to the corresponding bay on a host at another site.
4. If necessary, the shipping server on the receiving host sends the packet to its next destination.

Directories for Packets

Each storage class has storage bays and return bays, which are directories that hold packets. Storage bays are used for normal shipping operations, and return bays are used for packets that could not be delivered successfully.

Each storage bay and return bay directory contains two subdirectories, incoming and outgoing, which hold the packets and their corresponding shipping order files. Shipping operations look in these directories for packets.

Note: On Windows, the amount of available space on the disk partition where the bays are located must be at least twice the size of the largest packet that will be stored in the 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.

When you install MultiSite on a host, the **–default** storage class is created, along with its storage and return bays. The storage bay is named ms_ship and the return bay is named ms_rtn. The incoming and outgoing directories in each bay are also created. When you use the MultiSite Control Panel (Windows) to create a new storage or return bay, the bay and its subdirectories are created. On UNIX, you must create the bays and their incoming and outgoing subdirectories and then specify the bays in the shipping.conf file.

Packet Transport

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 the shipping server, the file is transferred.

Store-and-Forward Issues

The following sections describe issues to 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. To verify TCP/IP access, use **rcp** on each sending host to copy a file to the receiving hosts or use store-and-forward to send a packet (see *Submitting Packets to Store-and-Forward* on page 71).

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** and **syncreplica** commands fail if they try 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 packet into multiple packets:

multitool mkreplica –export –maxsize 1g ...

multitool syncreplica –export –maxsize 500m ...

You can also specify maximum packet sizes in the shipping.conf file (UNIX) or MultiSite Control Panel (Windows).

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

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.

Before you use store-and-forward, verify that you have the appropriate disk space, and configure the shipping.conf file or the MultiSite Control Panel and create storage classes for packets.

For more information about 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 **mkreplica** and **syncreplica** 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 the shipping server to send the packet files to their destination host or to 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 the shipping server. The packet files are sent the next time the shipping server polls the bay. For information about running the shipping server automatically, see *Automated Synchronization* on page 105.

Differentiating Packets with Storage Classes

You can configure the store-and-forward facility to handle packets 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 and return bays; in this case, the shipping server uses the size of the packet to select one of the bays. Conversely, several storage classes can share one or more bays.

You can use multiple storage classes to segregate the packets for VOBs that belong to different groups. By adjusting the operating system permissions on the storage bay and return 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.

If you are using the store-and-forward facility to transport packets from VOB replicas and from ClearQuest database replicas, you must use different storage classes.

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

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 the 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), the shipping server generates an error message and log file entry and exits. You can set up a retry scheme to control its frequency:

- After successful transmission of a packet, the 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 in the Scheduled Jobs for ClearCase 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* for Rational ClearCase and *Automated Synchronization* on page 105.

Attempts to transmit an undelivered packet can continue indefinitely, through repeated invocations of the **shipping_server** command. However, you 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 on the sending host
- (Windows) A **Packet Expiration** value in the MultiSite Control Panel on the sending host

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

When the 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 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 72. 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.

Setting a Timeout Period for Unreachable Hosts

If the shipping server tries to send a packet to a target host and determines that the host is unreachable, it creates a file in the /var/adm/rational/clearcase/shipping/ms_downhost directory (UNIX) or the *ccase-home-dir*\var\shipping\ms_downhost directory (Windows). The name of the file is the name of the unreachable host.

If one of the following parameters is set, the shipping server checks the directory for target hosts during future shipping operations:

- (Windows) **Timeout for Unreachable Host (minutes)** value in the MultiSite Control Panel
- (UNIX) **DOWNHOST-TIMEOUT** setting in the shipping.conf file or the SHP_DOWNHOST_TIMEOUT_RETRY environment variable (If both parameters are set, the shipping server uses **DOWNHOST-TIMEOUT**.)

If the target host is found in the ms_downhost directory, and the difference between the current time and the last modification time of the file is less than the timeout setting on the shipping server host, the shipping server does not try to send packets to the target host. If the difference is equal to or greater than the timeout setting, the shipping server tries to send packets to the target host. If the timeout setting is not set, the shipping server attempts to send the packet to the target host. (Each attempt to send a packet to an unreachable host takes about 30 seconds.)

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.

Sending Files That Are Not Packets

You can use the store-and-forward facility to send any file 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**:

  /opt/rational/clearcase/etc/mkorder –data /usr/rptgen/brdcst.0702 –fship –copy boston_hub tokyo

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

  /opt/rational/clearcase/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.

Using Store-and-Forward Through a Firewall (UNIX only)

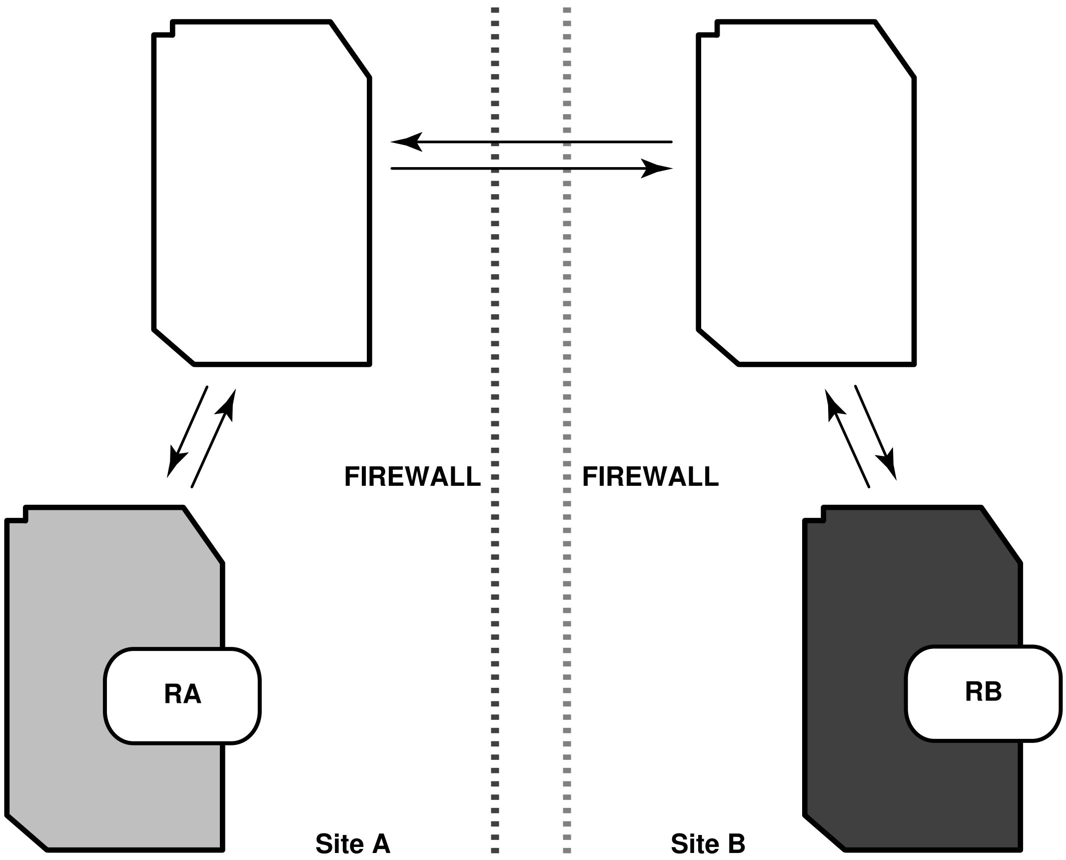
By default, the store-and-forward facility (the shipping server) cannot operate through a firewall. Passing through a firewall is usually accomplished by granting access to specific ports for certain IP addresses. Because the shipping server picks any available port number on the sending and receiving replica hosts to make the connection, there is no single port number (or even small range of port numbers) to which special access can be granted.

If your site uses a firewall, you can set up an "exposed host," a host that you configure to communicate through the firewall and on which you install the shipping server software. You configure the shipping servers on the replica hosts at your site to send packets to the exposed host, and the shipping server on the exposed host forwards the packets to hosts on the other side of the firewall. To maximize security on the exposed host, you must specify the range of port numbers that the shipping server can use.

Note: To enhance site security, we recommend that you install the shipping server on an exposed host only if other transport methods are unsuitable for your site. This method is not available for Windows. For information about other methods, see *File-Based Methods* on page 67.

Figure 18 is an example of an exposed host configuration. The exposed hosts communicate through the firewall. The store-and-forward software is installed on them, but ClearCase software is not installed on them. Rational ClearCase and MultiSite are installed on the replica server hosts (labeled **RA** and **RB**).

Figure 18 Store-and-Forward Configuration



Firewall Issues

Before installing the shipping server on an exposed host, consider the following issues:

- Shipping bays can be filled.

  Using the shipping server on an exposed 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:

  - Create all shipping bays in the local network on their own partitions, so that filling the bays does not degrade system performance.
  - Install the shipping server only on machines that need it: servers with replicas and machines used by administrators.

- Packets are susceptible to snooping.

  In normal update packets, information is not encoded. Therefore, anyone shipping packets across an unsecured network must encrypt the packets. Also, the format of an 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 exposed host that runs the shipping server must not run other ClearCase servers.

Configuring Your Firewall to Limit Access

We recommend that you specify the ports to which programs can connect and the IP addresses that are allowed to access the firewall. Limiting the allowed port numbers and IP addresses limits the possibility that unauthorized machines can breach the firewall.

You must allow access to the following ports on the exposed host:

- TCP port 371 (**albd_server** port)
- The range of ports that you specified with the CLEARCASE_MIN_PORT and CLEARCASE_MAX_PORT environment variables (see *Controlling Ports Used by albd_server and shipping_server* on page 79)

You must allow access through the firewall for IP addresses of hosts that send packets through the firewall to the exposed host at your site.

For information about configuring your firewall, see the documentation for your firewall.

Installing the Shipping Server on an Exposed Host

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 **ClearCase MultiSite Shipping Server-only Installation** option. Do not install ClearCase on the exposed host.

On Windows, there is no option to install only the shipping server software on an exposed host.

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 the shipping server can allocate for communication purposes. When the shipping 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 shipping server sleeps and tries the ports again.

When the shipping server on the sending host 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, the shipping server tries UDP. Therefore, if you have TCP connectivity, you do not have to enable UDP or open UDP ports on the exposed host.

Running an individual shipping server does not require more than two ports at a time. When there are multiple requests to be sent, the 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.

Specifying 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.

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 exposed host. For more information, see the **shipping.conf** reference page.
- The clearcase script on the exposed host:

  a Edit the file *ccase-home-dir*/etc/clearcase.

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
```

Checklist for Using Store-and-Forward Through a Firewall

This checklist summarizes the steps you must follow to use store-and-forward through a firewall.

1. Determine the port ranges that the shipping server can use and the IP addresses of the hosts that will send packets to your site's exposed host.
2. Configure your firewall to limit the allowed port numbers and IP addresses. Remember that you must allow access to TCP port 371 in addition to the port ranges.
3. Install the shipping server software on the exposed host.
4. Set the CLEARCASE_MIN_PORT and CLEARCASE_MAX_PORT environment variables.
5. On each replica server host at your site, specify the exposed host as the next-hop host for packets sent to other sites. For example, your company has three sites (SiteA, SiteB, SiteC), each with one exposed host running the shipping server (**SSA**, **SSB**, **SSC**), and three replica server hosts.

   On UNIX, edit the shipping.conf file and add **ROUTE** options. For example, on each replica server host at SiteA:

   ```
   ROUTE SSA SiteB_host1 SiteB_host2 SiteB_host3 SiteC_host1
   SiteC_host2 SiteC_host3
   ```

   On Windows, open the MultiSite Control Panel and set the appropriate values in the Routing Information section. For example, on each replica server host at SiteA, the **Next Routing Hop** is **SSA** and the **Destination Hostnames** are **SiteB_host1**, **SiteB_host2**, **SiteB_host3**, **SiteC_host1**, **SiteC_host2**, and **SiteC_host3**.
6. On the exposed host, edit the shipping.conf file and add **ROUTE** options for the next destination of the packets.

Using the same example as in Step 5, on the exposed host at SiteA, you add the following **ROUTE** options to the shipping.conf file:

```
ROUTE SSB SiteB_host1 SiteB_host2 SiteB_host3
ROUTE SSC SiteC_host1 SiteC_host2 SiteC_host3
```

On the exposed host at SiteB, you add the following **ROUTE** options to the shipping.conf file:

```
ROUTE SSA SiteA_host1 SiteA_host2 SiteA_host3
ROUTE SSC SiteC_host1 SiteC_host2 SiteC_host3
```

On the exposed host at SiteC, you add the following **ROUTE** options to the shipping.conf file:

```
ROUTE SSA SiteA_host1 SiteA_host2 SiteA_host3
ROUTE SSB SiteB_host1 SiteB_host2 SiteB_host3
```

Feature Levels 6

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

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 you 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 at sites running later versions of ClearCase from using new features that are not meaningful to replicas on hosts running earlier versions.

Each version of ClearCase, each VOB family, and each replica has a 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. For example:
  - In a VOB family, two replicas are at feature level **1** and the third replica is at feature level **2**. The VOB family feature level must be equal to or less than **1**.
  - In another VOB family, all replicas in the family are at feature level **2**. The VOB family feature level must be equal to or less than **2**.

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 85.

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.

Raising the Replica Feature Level

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

- The replica must be self-mastering.
- 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.

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, you must transfer mastership of the VOB object:

   a Change mastership of the VOB object:

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

   b Export an update packet to the replica whose feature level you want to raise:

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

   c 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 132.

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 to inform them about the feature level change.

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

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*.
- 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*.

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:

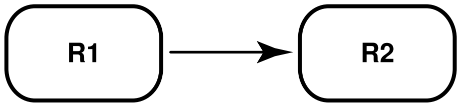
cleartool chflevel –family *feature-level* **vob:***vob-tag*

2 Export update packets to all other replicas in the VOB family to inform them about the feature level change.

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.

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: 4
  ...
  ```

- On Windows, start 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-02.14:19:03 by Susan Goechs (susan.user@minuteman)
  master replica: boston_hub@/vobs/dev
  replica name: boston_hub
  VOB family feature level: 4
...
```

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
```

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.
```

Replication and Synchronization

Creating VOB Replicas

7

This chapter describes how to plan and create VOB replicas. Before creating a replica, you must make decisions about branching, mastership, identities and permissions preservation, and method of packet delivery. Be sure to read *ClearCase Use Model* on page 34 and *MultiSite Use Model* on page 37.

Overview of Replica Creation

You use this three-phase procedure to create a new replica:

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

The procedure is similar for different methods of packet delivery and for different platforms. The example in this chapter assumes that you have a high-speed connection between hosts, and that 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 a 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 101.

Timing of Replica Creation

During the export phase, the **mkreplica** 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.

Replica-Creation Scenario for a VOB

For the example in this section, the company's 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 company uses the following 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 type 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 type. The San Francisco replica masters a branch type named **sanfran_main**, as well as any additional branch types 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 preserve identities. Because the administrators want to preserve changes to permissions, they decide to make the replicas permissions 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 VOB 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 who use the original VOB cannot access it 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 the appropriate 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 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 in the **mkreplica** command 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 the **cleartool space** command:

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 combined size of the VOB database and source pool is 1.62 GB, so 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 **mkreplica** command. For information about restrictions on the command, see the **mkreplica** reference page.

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

```
MINUTEMAN% multitool mkreplica –export –workdir /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
/opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_1
6-Aug-00.09.49.36_14075_1
- shipping order file is
/var/adm/rational/clearcase/shipping/ms_ship/outgoing/sh_o_repl_bos
ton_hub_16-Aug-00.09.49.36_14075_1
Dumping database...
...
Dumper done.
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_1
6-Aug-00.09.49.36_14075_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** command 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 "original" to "boston_hub"."
15-Aug.14:19   susan     make attribute "FeatureLevel" on replica
"boston_hub"
"Added attribute "FeatureLevel" with value 2."
...
```

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 replica.

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**, 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% /opt/rational/clearcase/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 running **lspacket** 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:
/opt/rational/clearcase/shipping/ms_ship/incoming/repl_boston_hub_1
6-Aug-00.09.49.36_14075_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 the import phase of replica creation:

- This replica is permissions 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 `Should I create this replica?` prompt during replica creation by specifying the **–vreplica** option with the new replica's name. This example does not specify that option.
- If you create a permissions-preserving replica, you can bypass the prompt during replica creation by specifying the **–nprompt** option with the **–perms_preserve** option. 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 –perms_preserve –work /tmp/wk
–tag /vobs/dev  –public –password 1234xyz –vob /vobstg/dev.vbs
/opt/rational/clearcase/shipping/ms_ship/incoming/repl_boston_hub_16-Aug-00
.09.49.36_14075_1
multitool: Warning: In a permissions-preserving replica, cleartool
protect operations will fail on client machines running ClearCase
versions associated with feature level 3 or lower.
Should I create a permissions-preserving replica? [no] yes
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
/opt/rational/clearcase/shipping/ms_ship/incoming/repl_boston_hub_1
6-Aug-00.09.49.36_14075_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 (If new replica preserves identities and permissions) Make sure that the owner, group, and group list of the new VOB are the same as the owner, group, and group list of the other identities- and permissions-preserving replicas in the VOB family. To change these values, run the **cleartool protectvob** command on the new VOB replica.

14 (If new replica preserves identities and permissions or permissions only) Send an update packet to all other replicas in the VOB family.

If you create a preserving replica, inform other replicas in the VOB family of this property. For example, **sanfran_hub** preserves permissions, so the San Francisco administrator enters this command:

`GOLDENGATE%` **multitool syncreplica –export –c "permissions preserving" –fship boston_hub**
...

15 Make the new replica self-mastering. See *Transferring Mastership of a Replica Object* on page 132 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.

16 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 4 sanfran_hub@/vobs/dev
Replica feature level raised to 4.
```

17 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
...
```

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

The San Francisco developers use branches of type **sanfran_main**.

```
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**), which is mastered by replica **sanfran_hub**, and label types (**SANFRAN_BASE** and **V1.0**), which are mastered by replica **boston_hub**. The San Francisco developers cannot create instances of the **SANFRAN_BASE** label type or move any existing **SANFRAN_BASE** labels, but there is no reason for them to do so.

19 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
...
```

20 (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 for a VOB Replica* on page 120.

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".
```

21 Begin development.

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

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 site for interoperability between UNIX and Windows, see Chapter 12, *Using MultiSite for VOB Backup and Interoperability*.

If your sites do not have an IP connection, you must use electronic mail, CDs, 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.

Synchronizing Replicas

8

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

1 Export: At one replica, a **syncreplica** (synchronize replica) command is invoked with the **–export** option. This creates a packet of data.

2 Transport: The packet is sent to one or more other replicas.

3 Import: At the other replicas, 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.

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 assumption 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 is unreadable at the sibling replica), you must adjust the epoch numbers at the sending replica to enable the lost data to be resent. For more information, see Chapter 13, *Troubleshooting MultiSite Operations*.

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**).

   The following example uses the **–fship** option to send the packet immediately:

   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-02.14.35.49_2468_1
    - shipping order file is C:\Program
   Files\Rational\ClearCase\var\shipping
   \ms_ship\outgoing\sh_o_sync_bangalore_30-Oct-02.14.35.49_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-02.14.35.49_2468_1
   ```

   The following 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, CDs, or tape.

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

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

Transport Phase

If you did not use the **–fship** option, send the packets:

- If you used **syncreplica –export –ship**, invoke **shipping_server** in either of the following ways:

  shipping_server –poll
  shipping_server *shipping-order-pathname*

- If you did not use **–fship** or **–ship**, send the packets using electronic mail, regular mail, or your preferred delivery method.

Import Phase

2. (If you used diskettes, CDs, tape, or electronic mail) Copy the packet files into a directory on the receiving replica's host.

3 Use the **lspacket** command to verify that the packet has arrived.

multitool lspacket

```
Packet is:
/opt/rational/clearcase/shipping/ms_ship/incoming/sync_bangalore_30
-Oct-02.14.35.49_2468_1
Packet type: Update
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:
boston_hub
The packet sequence number is 1
```

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

```
Applied sync. packet
/opt/rational/clearcase/shipping/ms_ship/incoming/sync_bangalore_30
-Oct-02.14.35.49_2468_1 to VOB /net/minuteman/vobstg/dev.vbs
```

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
```

Automated Synchronization

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

- **Export phase**. A script exports update packets from one or more replicas on a host to one or more siblings.
- **Transport phase**. The store-and-forward facility sends packets between sites. You can invoke store-and-forward as part of the export phase, or automate packet transport separately.
- **Import phase**. A receipt handler imports packets when they are 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 scripts that are shipped with Rational ClearCase MultiSite at any time and with any frequency, and you can vary the strategy for different families by using multiple jobs.

By default, the 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 scripts log their activity to files in the /var/adm/rational/clearcase/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 108), 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 **schedule** reference page in the *Command Reference* and the scheduler information in 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 about 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
```

For information about running **shipping_server**, see *Transport Phase*.

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. The shipping server is run every half hour from midnight to 04:00 local time:

```
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 14.

Table 14 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.

Table 14 Import Methods

Import method	Description	Advantages	Disadvantages
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 and receipt handlers 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
/opt/rational/clearcase/config/scheduler/tasks/sync_receive
```

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

Defining Receipt Handlers 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, edit the copy, 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 about the script's 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
```

Listing Synchronization History

The **lshistory** command and the History Browser list the history of a VOB replica, including synchronization information. For more information, see *Listing the Synchronization History of a VOB Replica* on page 116.

Synchronizing VOB Replicas More Efficiently

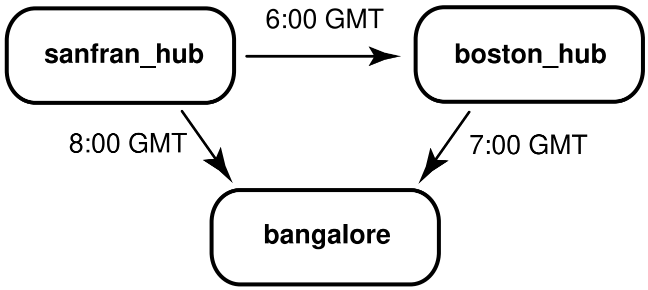
You can configure synchronization updates to send only the necessary operations to another replica. Although sending an operation multiple times does no harm, packet creation and transmission is more efficient if you exclude operations that have already been imported at the receiving replica.

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 19. All replicas use receipt handlers, so incoming packets are imported immediately. First, replica **sanfran_hub** sends a packet to replica **boston_hub**. Next, **boston_hub** sends a packet to **bangalore**. This packet includes operations from **sanfran_hub**.

Figure 19 Partial Synchronization Export Pattern and Schedule



At 8:00 GMT, **sanfran_hub** sends a packet to **bangalore**. This packet contains operations originating at **sanfran_hub** that **bangalore** has already received from **boston_hub**. In this case, use the command **chepoch –actual bangalore** at **sanfran_hub** before generating an update packet for **bangalore**. When you generate the packet, the operations already imported at **bangalore** are 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 15 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 four packets are 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**. You can use the **chepoch –actual –raise_only** command to avoid sending operations multiple times. With **–raise_only**, **chepoch** does not lower epoch numbers for sibling replicas.

MultiSite Management

Managing Replicas

9

This chapter describes how to manage existing replicas, including how to delete a replica. For information about creating a replica, see Chapter 7, *Creating VOB Replicas*. For information about enabling requests for mastership in a VOB replica, see Chapter 11, *Implementing Requests for Mastership*.

Displaying Properties of a VOB Replica

The **describe** command, which is available in **cleartool** and **multitool**, displays the properties of a VOB 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. Start 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**.

- At 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

Listing the Synchronization History of a VOB 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"
```

Changing Preservation Mode for a VOB Family

Any subset of replicas in a VOB family can preserve identities and permissions. Within this set 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 set has no immediate effect on the replica's objects. However, future changes to identities are propagated among all of the replicas in the VOB family that preserve identities and permissions. Future changes to permissions are propagated among all of the replicas that preserve identities and permissions or permissions only.

Note: On UNIX, preserving changes to identities across all sites is possible only if all sites support the same user-group accounts. On Windows, ownership modes (UIDs and GIDs) 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 preserve identities and permissions. You can preserve identities and permissions in a subset of replicas in the same domain. In a mixed environment, you cannot preserve identities and permissions on the entire set of replicas. For more information, see *Identities and Permissions Strategy for VOB Replicas* on page 39.

You can change a replica in the following ways:

- Identities and permissions preserving to nonpreserving or permissions preserving. For example, if a replica was created incorrectly as identities and permissions preserving, you may need to change it.
- Permissions preserving to identities and permissions preserving.
- Permissions preserving to nonpreserving.
- Nonpreserving to identities and permissions preserving. The replica will receive future changes to identities and permissions, but the original information is not restored.
- Nonpreserving to permissions preserving. The replica will receive future changes to permissions, but the original information is not restored.

To change a replica's preservation property:

1 At the master replica, change the replica property.

On UNIX or Windows, you can use the **chreplica** command.

To change from identities and permissions preserving or permissions preserving to nonpreserving:

multitool chreplica –npreserve boston_hub@/vobs/dev

```
Updated replica information for "boston_hub".
```

To change from identities and permissions preserving to permissions preserving:

multitool chreplica –perms_preserve boston_hub@/vobs/dev

```
multitool: Warning: Although replica now preserves only permissions,
existing objects may still reflect previous identities- and
permissions-preserving state.
multitool: Warning: In a permissions-preserving replica, cleartool
protect operations will fail on client machines running ClearCase
versions associated with feature level 3 or lower.
Updated replica information for "boston_hub".
```

To change from permissions preserving to identities and permissions preserving:

multitool chreplica –preserve boston_hub@/vobs/dev

```
multitool: Warning: Although replica now preserves identities and
permissions, existing objects may still reflect previous
permissions-preserving state.
Updated replica information for "boston_hub".
```

To change from nonpreserving to identities and permissions preserving:

multitool chreplica –preserve boston_hub@/vobs/dev

```
multitool: Warning: Although replica now preserves identities and
permissions, existing objects may still reflect previous
non-preserving state.
Updated replica information for "boston_hub".
```

To change from nonpreserving to permissions preserving:

multitool chreplica –perms_preserve boston_hub@/vobs/dev

```
multitool: Warning: Although replica now preserves permissions,
existing objects may still reflect previous non-preserving state.
multitool: Warning: In a permissions-preserving replica, cleartool
protect operations will fail on client machines running ClearCase
versions associated with feature level 3 or lower.
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 VOB Replica* on page 115.

b Select the appropriate preservation mode.

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

If you change the replica to permissions preserving or identities and permissions preserving, the appropriate warning is displayed (see the **chreplica** output above).

For restrictions, see the **chreplica** reference page.

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 add the missing groups to the importing VOB replica, and then try the import again.

If the import succeeds, you can use the **protectvob** command to undo the group changes you made.

4 (If the replica was changed to identities- and permissions-preserving) At the changed replica, use the **cleartool protectvob** command to set the owner, group, and group list of the VOB to be the same as the owner, group, and group list of the other identities- and permissions-preserving replicas in the VOB family.

5 (If the replica was changed to nonpreserving) 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.

For example, on UNIX, the following command finds all file and directory elements in **/vobs/dev** and executes the **protect** command:

cleartool find /vobs/dev -all -type fd –exec ’cleartool protect –chown vobowner –chgrp vobgrp $CLEARCASE_PN’

On Windows, the following command finds all file and directory elements in **\dev** and executes the **protect** command:

cleartool find \dev -all -type fd –exec "cleartool protect –chown vobowner –chgrp vobgrp %CLEARCASE_PN%"

Changing the Host Name for a VOB Replica

When you move a VOB 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 VOB Replica* on page 115.

2 On the **General** tab, type the new host name in the **Host** box.

3 Click **OK** or **Apply**.

Setting the Connectivity Property for a VOB Replica

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".
```

Use the **describe** command to verify that the property is set:

multitool describe replica:sanfran_hub
```
replica "sanfran-hub"
...
  connectivity: connected
```

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

Renaming a VOB 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 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 VOB Replica* on page 115.
2. Enter a new value in the **Name** box.
3. Click **OK** or **Apply**.

Moving a VOB Replica

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

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

- Rational ClearCase MultiSite must be 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 that you moved.
- After moving the VOB replica, export update packets to all sibling replicas.

Changing Mastership of a VOB Replica

For information about changing mastership of a VOB replica, see Chapter 10, *Managing Mastership*.

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 family.

When you remove a replica, the replicas in its family stop tracking epoch numbers for that replica. Removing a replica does not delete the VOB database.

Removing a replica requires two synchronization cycles: one to transfer mastership of all of the replica's objects to another replica, and one to inform all other replicas that the removed replica is no longer participating in the update process. Because this information can be communicated only through the synchronization process, you cannot remove a replica at its own site, because doing so prevents the replica from creating update packets.

After a replica is removed from a family, it no longer participates in synchronization activities and MultiSite information is not tracked. The replica no longer updates its oplog, and you cannot transfer mastership of any object in that replica.

Note: If a VOB replica is deleted mistakenly and you want to restore it from backup, see *Restoring and Replacing VOB Replicas* on page 198. If a VOB replica's storage directory is lost and there is no backup, see *Cleaning Up After Accidental Deletion of a Replica* on page 204.

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 decommissioned replica, transfer mastership of all objects it masters to another replica. If the decommissioned replica 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 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 138. 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 After Accidental Deletion of a Replica* on page 204.

3 Export and send an update packet from the decommissioned replica.

The decommissioned replica must send its final changes, including checkins and mastership changes, to the replica receiving mastership. The decommissioned replica 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
Generating synchronization packet c:\Program
Files\Rational\ClearCase\var
\shipping\ms_ship\outgoing\sync_tokyo_23-Dec-02.09.33.02_3447_1
 - shipping order file is c:\Program
Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\sh_o_sync_to
kyo_23-Dec-02.09.33.02_3447_1
Attempting to forward/deliver generated packets...
 -- Forwarded/delivered packet c:\Program
Files\Rational\ClearCase\var
\shipping\ms_ship\sync_tokyo_23-Dec-02.09.33.02_3447_1
```

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

```
GOLDENGATE% multitool syncreplica –import –receive
Applied sync. packet
/opt/rational/clearcase/shipping/ms_ship/incoming/sync_tokyo_23-Dec
-02.09.33.02_3447_1
to VOB /net/goldengate/vobstg/tests.vbs
```

5 At the master replica, remove the decommissioned replica.

```
GOLDENGATE% multitool rmreplica tokyo@/vobs/tests
Deleted replica "tokyo".
```

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
Generating synchronization packet ...
```

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.

10 Managing Mastership

This chapter describes how to manage the mastership of objects in a replica. Before reading this chapter, you should read the information in *Enabling Independent Development: Mastership* on page 5.

Mastership Commands for VOB Objects

The following commands are used to manage mastership of VOB objects:

- **chmaster** (**cleartool** and **multitool**)
- **describe** (**cleartool** and **multitool**)
- **lsmaster** (**cleartool** and **multitool**)
- **mkelem –master** (**cleartool** only)
- **mkdir –master** (**cleartool** only)
- **reqmaster** (**cleartool** and **multitool**)

For more information about the commands, see their reference pages in this manual or in the *Command Reference* for Rational ClearCase.

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. These interfaces are described in Chapter 11, *Implementing Requests for Mastership*.

Displaying Mastership Information for VOB Objects

The following sections describe how to list the master replica of an object, list all objects mastered by a particular replica, list the mastership changes for an object, and display settings for mastership requests.

Listing an Object's Master Replica

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

Method	Description
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 about 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
  ```

Listing Objects Mastered by a Replica

The **lsmaster** command lists the objects mastered by a VOB 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**), using information at the current replica:

  multitool lsmaster boston_hub@/vobs/dev

- To list all label types mastered by replica **sanfran_hub**, using information at 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.

Listing the History of Mastership Changes for an Object

The **lshistory –minor** command and the History Browser (with the **Include minor events** toolbar icon 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"."
...
```

Displaying Mastership Request Settings

The settings for mastership requests controls whether developers at other replicas 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 about mastership requests, see Chapter 11, *Implementing Requests for Mastership*.

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 things:

- Use the command **cleartool mkelem –master**.
- Use the command **cleartool mkdir –master**.
- (Windows; file elements only) 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
element * /main/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 –nc 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-Sep-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 "syntax file" cmdsyn.c

```
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@@"
```

Changing Mastership of VOB Objects

When you create an object in a replicated VOB, your current replica is the new object's master replica. To transfer mastership of the object to another replica, use 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 replica at that 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 122.)

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, before the packet is imported at the **bangalore** replica:

- 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 at any replica 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 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 139.
- 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 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 Help.

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 if the replica masters the target object. (For more information, see *Type Object Mastership* on page 16; there are additional restrictions if the type is a global type.)

Note: If you change mastership of a branch type to another replica, mastership is not changed for explicitly mastered branches of that type, even if the same replica masters the branch type and the branch. Mastership is changed for branches of that type with default mastership. To change explicitly mastered branches to have default mastership, see *Removing Explicit Mastership of a Branch* on page 136.

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 and send an update packet to the new master replica:

MINUTEMAN% **multitool syncreplica –export –fship sanfran_hub@/vobs/dev**

```
Generating synchronization packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_2
6-Aug-02.18.15.57_7430_1
- shipping order file is
/opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_sync_boston_
hub_26-Aug-02.18.15.57_7430_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_2
6-Aug-02.18.15.57_7430_1
```

4 At the new master replica, import the packet:

BAGUETTE% **multitool syncreplica –import –receive**

```
Applied sync. packet
/opt/rational/clearcase/shipping/ms_ship/incoming/sync_boston_hub_2
6-Aug-02.18.15.57_7430_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-02.14:20:26 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 administrator for Rational ClearCase MultiSite, self-mastering replicas simplify replica maintenance because each one 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 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
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_1
   6-Aug-00.16.15.57_6389_1
   - shipping order file is
   /opt/rational/clearcase/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
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_1
   6-Aug-00.16.15.57_6389_1
   ```

4 At the new master replica, import the packet:

   GOLDENGATE% **multitool syncreplica –import –receive**
   ```
   Applied sync. packet
   /opt/rational/clearcase/shipping/ms_ship/incoming/sync_boston_hub_1
   6-Aug-00.16.15.57_6389_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. If you need to lock the VOB with the obsolete option, you must do it at the VOB's master replica.

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 and send an update packet to the new master replica:

   ```
   MINUTEMAN% multitool syncreplica –export –fship sanfran_hub@/vobs/dev
   Generating synchronization packet
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_2
   0-Sep-02.17.35.45_22513_1
   - shipping order file is
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_sync_boston_
   hub_20-Sep-02.17.35.45_22513_1
   Attempting to forward/deliver generated packets...
   -- Forwarded/delivered packet
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_2
   0-Sep-02.17.35.45_22513_1
   ```

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

   ```
   GOLDENGATE% multitool syncreplica –import –receive
   Applied sync. packet
   /opt/rational/clearcase/shipping/ms_ship/incoming/sync_boston_hub_2
   0-Sep-02.17.35.45_22513_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 identities or permissions on the element (for replicas that preserve identities and permissions).
- Changing permissions on the element (for replicas that preserve permissions).
- Locking the element with the obsolete option.
- Removing the element.

To transfer mastership of an element to another replica:

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 and send an update packet to the new master replica:

```
MINUTEMAN% multitool syncreplica –export –fship bangalore@/vobs/dev
Generating synchronization packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_0
7-Dec-02.18.15.57_5978_1
- shipping order file is
/opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_sync_boston_
hub_07-Dec-02.18.15.57_5978_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_0
7-Dec-02.18.15.57_5978_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-02.18.15.57_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 11, *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:

   `MINUTEMAN%` **multitool syncreplica –export –fship bangalore@/vobs/dev**

   ```
   Generating synchronization packet
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_1
   0-Dec-02.18.15.57_3056_1
   - shipping order file is
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_sync_boston_
   hub_10-Dec-02.18.15.57_3056_1
   Attempting to forward/deliver generated packets...
   -- Forwarded/delivered packet
   /opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_1
   0-Dec-02.18.15.57_3056_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-02.18.15.57_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 15, a branch can have default or explicit mastership. After you follow the steps in *Transferring Branch Mastership* on page 136, 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_bangalo
re_11-Dec-02.18.15.57_9476_1
- shipping order file is
/opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_sync_bangalo
re_11-Dec-02.18.15.57_9476_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_bangalore_11
-Dec-02.18.15.57_9476_1
```

4 At the replica that masters the branch type, import the packet:

```
MINUTEMAN% multitool syncreplica –import –receive
Applied sync. packet
/opt/rational/clearcase/shipping/ms_ship/incoming/sync_bangalore_11
-Dec-02.18.15.57_9476_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 instructions, see *Deleting a Replica* on page 122.

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:

1. At your replica, send an update packet to the new master replica.
2. At the new master replica:
   a. Import the packet.
   b. Change mastership back to your replica.
   c. Export and send an update packet to your replica.
3. At your replica, import the packet.

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, with some restrictions if you are using global types (see *Additional Restrictions for Shared Global Types* on page 17).

For more information about type objects, see *Type Object Mastership* on page 16.

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".
```

Determining 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 only 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 a Type Object from Unshared to Shared

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** and the Bangalore project manager also needs to use it, you can convert the type to shared.

Note: You cannot convert a shared type object to unshared. To restrict instance creation of a type to one replica, you must 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 Help.

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***object***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 and send an update packet to the other replicas that must use the type:

multitool syncreplica –export –fship bangalore boston_hub

```
...
```

4 At the receiving replicas, import the update packet:

multitool syncreplica –import –receive

```
...
```

5 At the receiving replicas, confirm that the type object is shared:

cleartool describe –fmt "%n\t%[type_mastership]p\n" attype:TESTED_BY@/vobs/dev

```
TESTED_BY     shared
```

Implementing Requests for Mastership

11

To support development of VOB 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 about branch mastership in Chapter 1, *Introduction to MultiSite*, and *Branching and Mastership* on page 34.

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 (even if the developer is on the **–nusers** list).

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 all of the branch type's instances that have default mastership. If requests are denied at the replica or branch type level, or for any instances that have default mastership, the request fails.

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 (even if the developer is on the **–nusers** list).

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 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. Developers using that replica 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 for the replica or when an administrator imports the packet manually.

Requirements and Recommendations

In order for you 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 about feature levels, see Chapter 6, *Feature Levels*.
- The replica hosts have high-speed connections (LAN, WAN, T1).

  A request for mastership makes remote procedure calls (RPCs) directly to remote servers and fails if the hosts 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 by a replica at a site with a firewall.

- Each replica masters its own replica object.

  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 132.

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 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 replicas 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 replica 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.

- You use receipt handlers to import packets at each replica host.

  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 *Defining Receipt Handlers on UNIX* on page 109 and *Defining Receipt Handlers on Windows* on page 109.

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 must be 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 graphical interface for Rational ClearCase MultiSite 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 the ClearCase Administration Console to hide request for mastership features:

1. Navigate to the **ClearCase Registry** node in the ClearCase Administration Console.
2. Click **Action > Properties**.
3. Click **Help** and follow the instructions.

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.

Prerequisites

1. Verify that the replica is self-mastering. See *Transferring Mastership of a Replica Object* on page 132.
2. Verify that the feature level of each replica in the VOB family is the correct value, and that the VOB family's feature level is the correct value. For instructions, see Chapter 6, *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 and no white space. 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
```

Denying 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.

Enabling 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 hierarchy, you enable requests for mastership in the VOB replicas linked to the administrative VOB. 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; 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.

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/rational/clearcase/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/rational/clearcase/config/rfm_shipping.conf does not exist, the installation creates it by copying the template file. If /var/adm/rational/clearcase/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.

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.

Mastership request settings are not replicated, so the **describe** command and the **Mastership** tab display the current replica's settings. On Windows, the Request Mastership dialog box has an option to display the settings at the master replica.

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
  permissions: 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 branch type
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/plans/v3.0.doc@@/main

```
denied
```

Troubleshooting Mastership Requests

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 154 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

```
main chmaster
Reqmaster changed master replica from "sanfran_hub" to "boston_hub".
Requester: user "PURPLEDOC\susan" in domain "PURPLEDOC" on host
"minuteman"
main chmaster
Reqmaster changed master replica from "tokyo" to "sanfran_hub".
Requester: user "PURPLEDOC\jcole" in domain "PURPLEDOC" on host
"goldengate"
main chmaster
Reqmaster changed master replica from "bangalore" to "tokyo".
Requester: user "PURPLEDOC\masako" in domain "PURPLEDOC" on host
"shinjuku"
...
```

Status Messages

Table 15 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 16 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 15 Error Messages from Mastership Request Management Operations (Part 1 of 3)

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 and syntax 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)

Table 15 Error Messages from Mastership Request Management Operations (Part 2 of 3)

Message	Meaning of message and action to take
`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.`	
`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. Then 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 132.
`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.

Table 15 Error Messages from Mastership Request Management Operations (Part 3 of 3)

Message	Meaning of message and action to take
`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 16 Error Messages from Mastership Requests (Part 1 of 3)

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 and syntax of the object selector.
`Incompatible versions of ClearCase software and/or databases`	Your client host is running a later major version of ClearCase than the replica that masters the branch or branch type. Regular ClearCase client/server compatibility rules apply to mastership requests.
`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. **Note**: The **reqmaster** command does not check the **–nusers** list associated with the lock, so a request for mastership will fail even if you are on the **–nusers** list.
`Requests are denied for all objects mastered by the sibling replica.`	Mastership requests are not enabled for the replica. Ask the administrator of the master replica of the branch or branch type to enable mastership requests at the replica level.

Table 16 Error Messages from Mastership Requests (Part 2 of 3)

Message	Meaning of message and action to take
`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.`	You 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)
`Requests for mastership of UCM objects are not supported.`	You cannot request mastership of a branch or branch type associated with a UCM stream.
`The object is not a branch or a branch type.`	You 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 already mastered by replica '`*replica-selector*`'.`	Your current replica already masters the requested object.

Table 16 Error Messages from Mastership Requests (Part 3 of 3)

Message	Meaning of message and action to take
`The object was not found at the sibling replica. This may indicate that the replicas are not in sync.`	Your 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. ➤ Send update packets 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.`	Your 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.`	You are 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

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 in Boston 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 replicas at 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 **v3.0.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/plans/v3.0.doc@@/main /vobs/doc/plans/schedule.doc@@/main /vobs/doc/plans/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 config spec
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 config spec
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 config spec
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" 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 "update.fm".
```

2 She requests mastership of branch **update.fm@@\main**:

cleartool reqmaster –c "Tokyo needs mastership" update.fm@@\main

```
update.fm@@\main: Change of mastership at sibling replica
"boston_hub" was successful.
Mastership is in transit to the new master replica.
```

3 Periodically, she retries the checkout or displays properties of the branch to determine whether mastership has been received. 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" update.fm@@\main

```
tokyo@\doc
```

cleartool checkout –c "new command options" update.fm

```
Checked out "update.fm" from version "\main\30".
```

Serial Development of a File That Can Be Merged

1 John, in San Francisco, needs to change a 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

```
/vobs/html/scripts/conv_fm.pl@@/main: Change of mastership at
sibling replica "boston_hub" was successful.
Mastership is in transit to the new master replica.
```

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".
```

Requesting Mastership of a Branch Type

The Boston developers have been using nonmastered checkouts to work on scripts, and their project leader decides that all the changes need to be checked in. In order to reduce the number of mastership requests for individual branches, the project leader requests mastership of the **main** branch type.

1 She requests mastership of the branch type:

cleartool reqmaster –c "merging party in Boston" brtype:main@/vobs/html

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

2 Periodically, she displays properties of the branch type to determine whether mastership has been received. After mastership is received at her replica, the **describe** command shows that her replica masters the branch type.

cleartool describe –fmt "%[master]p\n" brtype:main@/vobs/html

```
boston_hub@/vobs/html
```

All of the branches that had default mastership are now mastered by the Boston replica. If any of the branches were explicitly mastered by the San Francisco or Tokyo replicas, their mastership was not changed. Developers must request mastership for those specific branches.

The developers in Boston can perform necessary merges and check in their changes.

Using MultiSite for VOB Backup and Interoperability

12

This chapter describes how to use Rational ClearCase MultiSite to back up a VOB and to provide access to VOBs in a heterogeneous network.

Backing Up VOBs with MultiSite

There are two ways to use MultiSite as a backup strategy:

- Using a replica as a backup 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 VOB Replicas* on page 198.

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 Rational ClearCase license for each developer who accesses the replicated VOB.

Using a Backup Replica

To back up a VOB consistently, the ClearCase administrator must lock the VOB. However, many administrators 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 you 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 some objects in a VOB are not replicated. 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 ensure that 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, use the **describe trtype:** command to list details about each trigger, and then save that information somewhere outside the VOB.

- Nonobsolete locks

  As with triggers, you must record information about nonobsolete locks. You can write 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 replicas synchronize every 24 hours, but replicas with rapid development may synchronize every 15 minutes.

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 199, 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 with a different registry host.

Restoring a Replica from Backup

Use the procedure described in *Restoring a Replica from Backup* on page 199.

Using MultiSite for Interoperability

You can use multiple replicas in local area networks to provide native access to VOBs in a heterogeneous network. The following sections describe MultiSite support for multiple replicas in a LAN and give setup instructions.

Advantages and Disadvantages

Advantages of using MultiSite for interoperability:

- You do not need to purchase or maintain 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 replica 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 identities.

Restrictions on Multiple Replicas in a LAN

You must observe these restrictions when you create multiple replicas in a LAN:

- Do not register multiple replicas of a VOB family on a single registry host.

  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 in a VOB, there is no way to make the copy of the VOB unique. Creating and mounting 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.

Setting Up Multiple Replicas at One Site

This section describes the process of creating multiple 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 relocating VOB data, see the *Administrator's Guide* for Rational ClearCase.

If both replicas are on UNIX hosts or in the same Windows domain, they can preserve identities. 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
/opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_2
3-Dec-02.13.16.43_21767_1
 - shipping order file is
/opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_repl_boston_
hub_23-Dec-02.13.16.43_21767_1
Dumping database...
  . . .
Dumper done.
Attempting to forward/deliver generated packets...
  -- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_2
3-Dec-02.13.16.43_21767_1
```

2 On the Windows host:

multitool lspacket –short

```
c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\incoming\r
epl_boston_hub_23-Dec-02.13.16.43_21767_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_23-Dec-02.13.16.43_21767_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\r
epl_boston_hub_23-Dec-02.13.16.43_21767_1
...
```

Troubleshooting

Troubleshooting MultiSite Operations

13

This chapter describes common situations in which running a Rational ClearCase 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 problems, and other problems that you resolve by performing a recovery procedure.
- **Serious errors** are problems that may require assistance from Rational Customer 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.

For information about changing mastership, see Chapter 10, *Managing Mastership*. For information about mastership request errors, see Chapter 11, *Implementing Requests for Mastership*.

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/rational/clearcase/log/sync_logs (UNIX) *ccase-home-dir*\var\log (Windows)
Transport problems	shipping

Mastership request problems	msadm
Other errors	Command window Event Viewer (Windows)

  - ClearCase log files. If Rational 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.

- Install the latest product 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 view drive or 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 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 Customer Support:

  multitool –version
  multitool lsreplica –long
  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).

- You can list the history of exports and imports at your replica.

  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 the 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"
...
```

  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 the 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"
```

Replica Export Problems

If the **mkreplica –export** command finds that a replica with the specified name exists in the 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 replica. To complete the replica creation, use **shipping_server** to transfer the replica-creation packet.

Replica Import Problems

The following sections describe how to fix problems that occur during import of a replica-creation packet.

Permissions Problems

When you use the **–preserve** option with **mkreplica –import**, import of a replica-creation packet fails if the identity of the user importing the packet does not have rights to create source container files with particular groups. To fix this problem, do one of the following things:

- Add the missing group to the user's group list.
- Run **mkreplica –import** with the **–ignoreprot** option. With this option, the import will be completed even if protection failures occur. After the import is done, you must run **checkvob** to find and fix protection problems.

Conflict in Object Registry

A recoverable error occurs if the **mkreplica –import** command detects a conflict at the registry level because an entry exists in the object registry:

`Replica` *replica-name* `already exists`

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 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/rational/clearcase/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 Tag Registry

A recoverable error occurs if the **mkreplica –import** command detects a conflict at the registry level because an entry exists in the tag registry:

`Replica` *replica-name* `already exists`

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).

Synchronization Export Problems

This section describes problems that can occur during the export phase of synchronization.

Cannot Find Oplog Entry

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 about oplog entries, see *The Operation Log* on page 22 and *Scrubbing Parameters for Replicas* on page 52.)

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 you change the synchronization pattern for a family, replicas 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 is not reflected in **sydney**'s epoch number matrix. 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 *chepoch –actual Method*. Otherwise, use the procedure in *lsepoch and chepoch Method*.

chepoch –actual Method

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

lsepoch and chepoch Method

Proceed as follows:

1. At **buenosaires** (destination replica), run **lsepoch** to determine the actual state of **buenosaires**:

   multitool lsepoch buenosaires@/vobs/tests
   ```
   For VOB replica "/vobs/tests":
   Oplog IDs for row "buenosaires" (@ mardelplata):
   oid:ac93e6cf.14a311d5.bbcc.00:01:80:c0:4b:e7=4000     (buenosaires)
   oid:c6b8c9b0.038d11d1.b083.00:60:97:98:42:69=5927     (sydney)
   ```

2. 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)
   oid:ac93e6cf.14a311d5.bbcc.00:01:80:c0:4b:e7=4000
   oid:c6b8c9b0.038d11d1.b083.00:60:97:98:42:69=5927
   .
   Change oplog IDs in row "buenosaires" [no] yes
   Epoch row successfully set.
   ```

3. At **sydney**, enter the original **syncreplica –export** command.

   - If the command fails, **buenosaires** is in jeopardy. Have other replicas in the 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 201.

   - If the command succeeds and the packet is imported successfully at **buenosaires**, **buenosaires** is up to date.

Note: Have all administrators review their oplog scrubbing procedures. See *Scrubbing Parameters for Replicas* on page 52.

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 Entry* on page 178 do not work, contact Rational Customer 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.

Transport Problems

This section describes problems that can occur during the transport phase of synchronization.

Error Messages

The messages in Table 17 are generated by the **mkorder**, **mkreplica**, **shipping_server**, and **syncreplica** commands.

Table 17 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) The shipping server is not running as **root**, and *username* does not own the shipping order file.

Table 17 Shipping Error Messages (Part 2 of 2)

`cyclic delivery route detected to host` *hostname* `(via` *next-hop-hostname*`) for order` *shipping-order-pname*	The shipping order lists *next-hop-hostname* as a previous hop in the packet's delivery route. If the packet is sent to *next-hop-hostname* (which is specified in a **ROUTE** entry in the shipping.conf file or in the **Routing Information** section in the MultiSite Control Panel), it will eventually come back to the current host. Check the routing information on the hosts in the delivery path and fix any circular routes.
`file` *file-pname* `does not contain a valid shipping order`	The 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.
`giving up trying to return order` *shipping-order-pname* `to host` *hostname* `(original data file was` *file-pname*`)`	The 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 doesn't exist or is inaccessible.
`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, the shipping server fails, because it cannot determine where to deliver the packet. In this case, it writes error messages to its log file (UNIX) or the Windows Event Viewer.

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. 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 Customer Support.

Delivery Fails

Each time the shipping server cannot deliver a packet to a valid destination host, it logs error messages:

- (UNIX) In file /var/adm/rational/clearcase/log/shipping_server_log and writes a message to the terminal device, if there is one.
- (Windows) In the Windows Event Viewer. You can use the **cleartool getlog shipping** command to view **shipping_server** messages from the Event Viewer.

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 replica 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, the shipping server writes a message to the Windows Event Viewer.

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 CD, and mail the CD 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.

Synchronization Import Problems

This section describes problems that can occur during the import phase of synchronization.

Packets Accumulate in Incoming Storage Bay

A recoverable error occurs when an update packet is lost and is not applied to your replica. 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.

To verify that a packet is missing and determine which operations are needed:

1. Enter a **syncreplica –import –receive** command, which processes all incoming packets in the storage bay in the correct order. If **syncreplica** fails 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 media that you forgot to process or in packet files that were not processed because the shipping.conf file on UNIX or the MultiSite Control Panel on Windows specifies the wrong storage bay. If you locate the missing packets, do one of the following things:

- 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.)
- 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, see *Recovering from Lost Packets* on page 190.

Packet Is Not Applicable to Any Local 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 host-name property has not been updated with the **chreplica** command.

- To verify that the host-name property of a VOB replica 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 host name is incorrect, use the **chreplica** command to change it. At the master replica of the importing replica, enter a **chreplica** command:

multitool chreplica –c "*comment***" –host** *new-host*
replica:*importing-replica-name@VOB-tag*

For example:

multitool chreplica –c "change host name" –host brooklyn
replica:newyork@/vobs/tests
```
Updated replica information for "newyork".
```

Send an update packet to the other replicas in the family.

Read from Input Stream Fails

If a **syncreplica –import** command fails with a message like this one, the packet is corrupted:

```
Error: Read from input stream failed: No such file or directory
```

Delete the packet and ask the administrator at the sending replica to re-create the packet and send it again (see *Recovering from Lost Packets* on page 190). 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, run **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 records 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= 08/07/02 12:35:46 create_time= 08/07/02 12:35:46
16  event comment= "Destroyed replica "boston_hub".
```

This situation can occur if two replica hosts do not have the same patch level or if an 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 Customer Support.

Database Limit Is Exceeded

ClearCase versions 4.0 and later include support for a new VOB database schema. If you update one or more replicated VOBs in a family to the new schema (version 54), you do not have to update the other replicas in the VOB family immediately. However, you must update all replicas before one of the updated replicas exceeds the database limit of the previous schema (version 53). If you do not, replicas that have not been updated cannot import synchronization update packets from the updated replica.

When this type of import failure occurs, **syncreplica** output includes a VOB database error, and an error is written to the db log.

The **syncreplica** output includes an error like the following:

```
multitool: Error: Error from VOB database: ''\\vob.setup''.
```

The db log includes an error like the following:

```
09/20/96 10:40:49 db_server(19528): Error: DBMS error in
"../db__lock.c" line 79
*** db_VISTA database error -909 - file record limit exceeded
09/20/96 10:40:49 db_server(19528): Error: DBMS error
09/20/96 10:40:49 db_server(19528): Error: db_VISTA error -909
```

To fix this problem, you must convert all replicas in the family to schema version 54. To display the schema version for a VOB replica, use the **cleartool describe vob:***vob-tag* command. To display the schema version of the version of ClearCase installed on your computer, use the **cleartool –ver** command.

Replica Incarnation Is Old

The following error can occur during packet import:

`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 its 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-02.22:40:54UTC
   ...
   ```

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 190.
   - If the values are different, contact Rational Customer Support.

Warning on Receipt of Packet from Earlier MultiSite Version

Different versions of MultiSite have different packet protocols. When **multitool** with a newer protocol reads a packet with the older protocol, it prints this message:

`multitool: Warning: Version mismatch, software:`*new-protocol*`,`
`packet:`*old-protocol*

This message does not indicate a problem. It means one of the following things:

- The feature level of the family is lower than the feature level of the receiving replica.
- The feature level of the family is the same as the feature level of the sending and receiving replicas. However, when the sending replica created the update packet, it had not yet received a packet containing the information about the new family feature level.

Table 18 lists the packet protocols for MultiSite versions.

Table 18 MultiSite Releases and Packet Protocols

MultiSite version	Packet protocol
3.2, 3.2.1	1.2
4.0, 4.1, 4.2	3
2002.05.00	4
2003.06.00	5

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.
- 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 is destroyed or is 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 184.)

Lost Replica-Creation Packet

To recover a lost replica-creation packet:

1. At the replica where **mkreplica –export** was run, rename the new replica:

   cleartool rename bangalore@/vobs/dev bangalore-old@/vobs/dev
   ```
   Renamed replica from "bangalore" to "bangalore-old".
   ```

2. Reenter the original **mkreplica** command.

3. After synchronization between the exporting replica and the new replica is working, remove the renamed replica:

   multitool rmreplica bangalore-old@/vobs/dev
   ```
   Deleted replica "bangalore-old".
   ```

The following procedure is simpler, but the **rmreplica** command may take a long time if you have a large VOB:

1. At the replica where **mkreplica –export** was run, use **rmreplica** to remove the new replica.

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.

But, 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: Connected Method

1. At the sending replica, 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:

/opt/rational/clearcase/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: lsepoch and chepoch Method

1 At the receiving replica, 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 replica:

multitool chepoch sanfran_hub bangalore=950 boston_hub=1300 sanfran_hub=2000

```
Change oplog IDs in row "sanfran_hub" [no] yes
Epoch row successfully set.
```

Method 3: lshistory and chepoch Method

1 At the sending replica, 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 replica, use this output in a **chepoch** command:

multitool chepoch sanfran_hub bangalore=950 boston_hub=1300 sanfran_hub=2000

```
Change oplog IDs in row "sanfran_hub" [no] yes
Epoch row successfully set.
```

Method 4: lshistory and recoverpacket Method

1 At the replica 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 replica 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 replica, 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 replica.

```
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, and 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.

Preservation Mode

If two replicas preserve identities and permissions or permissions only, 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 must perform using **cleartool protectvob –add_group**.

syncreplica –import generates the following identities-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 a group was added to the sending replica's VOB group list, and someone created a new element in that group or reassigned an existing element to that group. Then, the change was sent 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 identities and permissions preserving.

If the replicas are intended to be identities and permissions 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 identities- and permissions-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 identities and permissions preserving, follow these steps:

1. Use the **multitool chreplica** command to change the receiving replica to permissions preserving or nonpreserving.

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

2. Import the packet.

   multitool syncreplica –import –receive
   ```
   Applied sync. packet
   /opt/rational/clearcase/shipping/ms_ship/incoming/sync_sanfran_hub_
   18-Jan-02.16.54.14_386_1 to VOB /net/minuteman/vobstg/dev.vbs
   ```

3. Change the status of the replicas.
   - If the sending replica should be nonpreserving or permissions-preserving, change it.
   - If you want to retain preservation of identities and permissions in the receiving replica, change it back to identities and permissions preserving.

4. Export update packets from the sending and receiving replicas to all siblings.

To avoid this problem in the future, use the procedure described in the section *Gathering Identities Information* on page 43.

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 branches 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, your replica may refuse to import this change because a trigger type in your replica 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 replica is useful only if the deleted type object exists, 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 **boston_hub** was the replica at which the type was created. **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 config specs, and renaming of replicas affects 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 administrators of the various replicas 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 replicas 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 replicas 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.)

Running epoch_watchdog

If a 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, operations 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 operations are recorded in the oplog starting with ID 601. These operations have the same ID as the operations 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 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 replicas on the host:

```
Job.Begin
  Job.Id: 20
  Job.Name: "epoch_watchdog"
  Job.Description.Begin:
  Run epoch_watchdog for each replica 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 scheduler information in the *Administrator's Guide* for Rational ClearCase.

Restoring and Replacing VOB 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 in the replica. 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 201.

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. Run the command to restore the replica:

   multitool restorereplica –invob *vob-selector*

   This places a special lock on the VOB object, which is separate from 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 are 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 190.)
- **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 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 family. For example, if you use 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 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 Customer Support advises you to follow these steps.

To replace a replica, use the following procedure (assume the **boston_hub** replica on host **minuteman** is to be replaced, and **sanfran_hub** and **bangalore** are the other replicas in the 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 203 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/rational/clearcase/shipping/ms_ship/incoming/sh_o_repl_sanfran_hub_18-May-02.15:50:00_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** mastered 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 replica 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 identify 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.

Cleaning Up After Accidental Deletion of a Replica

This situation is a more catastrophic variation of the problem described in *Restoring a Replica from Backup* on page 199: 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 122.

Perform this procedure in the replica that masters the deleted replica. (If the replica was its own master, perform this procedure in the replica that will master 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 mastered by the deleted replica. For example, if the **tokyo** replica is deleted, enter this command at the **sanfran_hub** replica:

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 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 family, to inform them of the mastership changes and the replica deletion.

multitool syncreplica –export ...

MultiSite Reference Pages

MultiSite Reference Pages 14

This section of the *Administrator's Guide* contains MultiSite reference pages.

apropos

Displays MultiSite command information

Applicability

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX

Synopsis

apr·opos *topic* ...

Description

This command displays information about MultiSite commands. Use **apropos** as you use the standard UNIX **whatis(1)** or **apropos(1)** command.

Restrictions

None.

Options and Arguments

topic ...

apropos searches for each *topic* character string in the standard MultiSite whatis file. The string can occur anywhere in the line.

Examples

- Search for lines with the string *epoch number* in the standard MultiSite whatis file.

  multitool apropos "epoch number"

  ```
  chepoch                   Changes epoch number estimates
  epoch_watchdog            Checks whether a replica's epoch numbers
  have rolled back when the replica is not in restoration mode
  ```

- Search for lines with the word *epoch* in the standard MultiSite whatis file.

  multitool apropos epoch

```
chepoch                  Changes epoch number estimates
epoch_watchdog           Checks whether a replica's epoch numbers
have rolled back when the replica is not in restoration mode
lsepoch                  Lists epoch information
recoverpacket            Resets epoch row table so changes in lost
packets are resent
```

Files

ccase-home-dir/doc/man/ms_whatis

See Also

In the *Command Reference*: **help**, **man**

chepoch

Changes epoch number estimates

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 a replica's epoch number estimates for other replicas. You cannot change a replica's own epoch numbers because they 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 *The Operation Log* on page 22. For descriptions of scenarios using **chepoch**, see *Cannot Find Oplog Entry* on page 178 and *Lost Update Packet* on page 191.

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. If you do not specify a *vob-selector*, the command uses the current VOB.

replica-selector

Specifies the replica whose estimated epoch numbers are 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 IDs in row "bangalore" [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 IDs in row "bangalore" [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 an 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 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 in 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 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
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 either of these conditions exist:

- There are reserved checkouts on that branch.
- There are unreserved checkouts on that branch made 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 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 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 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 *Transferring Mastership of a Stream* on page 138.

–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 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, but 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 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 specified by *brtype-selector*.

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-02.09.33.02_3447_1
  ...
  ```

  At the replica that masters the branch type:

  multitool syncreplica –import –receive

  ```
  Applied sync. packet
  /var/adm/rational/clearcase/shipping/ms_ship/incoming/sync_bangalor
  e_19-Aug-02.09.33.02_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 10, *Managing Mastership*

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** | **–per·ms_preserve** | **–npr·eserve**]
[**–isconn·ected** | **–nconn·ected**] *replica-selector*

Description

This command changes the properties of a replica. For more information, see *Changing the Host Name for a VOB Replica* on page 120, *Changing Preservation Mode for a VOB Family* on page 117, and *Setting the Connectivity Property for a VOB Replica* on page 120.

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 the options described in this section.

–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 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 Control Panel.

–pre·serve

Makes the specified replica identities- and permissions-preserving.

–per·ms_preserve

Makes the specified replica permissions-preserving.

–npr·eserve

Makes the specified replica nonpreserving.

–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 nonpreserving replica.

  multitool chreplica –npreserve tokyo@/vobs/doc

  ```
  Updated replica information for "tokyo".
  ```

- Mark replica **sydney** as not connected.

  multitool chreplica –nconnected sydney@\doc

  ```
  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 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 replica 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/rational/clearcase/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 the current number is lower than the logged number, **epoch_watchdog** checks to see if the replica is in restoration mode. If the replica is not being restored, **epoch_watchdog** attempts to lock the affected VOB, and optionally sends e-mail notification. You must specify e-mail addresses in the scheduled job. In this situation, you must contact Rational Support before unlocking the VOB or attempting any repair procedures.

Note: If you do not schedule **epoch_watchdog** to run frequently, the activity level in the replica can be high enough that a rollback may not be detected if **restorereplica** is not performed.

If an error occurs, **epoch_watchdog** creates an entry in /var/adm/rational/clearcase/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 form **vob:***VOB-tag*

Examples

For an example of the command, see *Running epoch_watchdog* on page 197.

Files

/var/adm/rational/clearcase/log/epoch_logs (UNIX)
/var/adm/rational/clearcase/log/error_log (UNIX)
ccase-home-dir\var\log\epoch_logs (Windows)

See Also

schedule (in the *Command Reference*)

lsepoch

Lists epoch information

Applicability

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

Synopsis

lsepoch [**–invob** *vob-selector* | [**–actual**] *replica-selector* ...]

Description

This command displays the epoch number matrix for a replica. 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 deleted replicas, 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 operations 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

Default

Displays the epoch number matrix of the replica containing the current working directory.

–invob *vob-selector*

Displays the epoch number matrix of the current replica in the 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 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 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, the output does not include the new 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 have the following values:

Values for ClearCase:

attype	**hlink**
branch	**hltype**
brtype	**lbtype**
delem (directory element)	**slink**
eltype	**vob**
felem (file element)	

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. For details about using this option, see the **fmt_ccase** reference page .

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

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- 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*

lspacket

Describes contents of a 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 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 one part of a logical packet with 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 that indicate the contents of the packet
- Recovery incarnation of the sending replica (an internal value used by MultiSite)
- Major and minor packet versions (internal values used by MultiSite)

–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

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- List a single replica-creation packet.

  multitool lspacket /opt/rational/clearcase/shipping/ms_ship/incoming/repl_boston_15-Aug-00.17.07.20_7865_1

```
Packet is:
/opt/rational/clearcase/shipping/ms_ship/incoming/repl_boston_15-Au
g-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 4, the minor packet version is 0
The packet sequence number is 1
```

See Also

mkreplica, **MultiSite Control Panel**, **syncreplica**, **shipping.conf**

lsreplica

Lists VOB replicas

Applicability

Product	Command type
ClearCase	cleartool subcommand
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 that contain 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

Lists 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. For details about using this report-writing facility, see the **fmt_ccase** reference page in the *Command Reference*.

–sib·lings

Lists the family members of the current replica, but does not list the current replica itself. Use this option in scripts that process only sibling replicas.

Specifying the VOB Family

Default

Lists family members of the replica containing the current working directory.

–invob *vob-selector*

Lists the replicas in the specified 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 in the 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

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- 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 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** *comment* | **–cq** | **–cqe** | **–nc**]
[**–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 file or any other file. The shipping order is used by the shipping server to send the file to one or more destinations.

mkorder submits to the shipping server a packet that was created with **mkreplica –out** or **syncreplica –out**. You can also use **mkorder** to resubmit 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 shipping server 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 it is delivered 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 change allows packets to be delivered to Windows machines, which do not allow colons in 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 shipping.conf file on UNIX or in the MultiSite Control Panel on Windows 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 shipping.conf file or the MultiSite Control Panel.

–shi·p –cop·y
–fsh·ip [–cop·y]

Creates a shipping order for *packet-pname*. Using **–fship** 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 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, the e-mail address of the administrator, and the notification program.

–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. This option overrides the expiration period specified for the storage class in the shipping.conf file (UNIX) or MultiSite Control Panel (Windows).

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-2002
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

–nc (no comment).

–c *comment* | **–cq** | **–cqe** | **–nc**

Specifies a comment to be placed in the shipping order. With **–c**, the comment string must be a single command-line token; typically, you must enclose it in double quotes. With **–cq** and **–cqe**, the command prompts you for a comment. With **–nc**, no comment is placed in the shipping order.

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 resides or is to be created.

Examples

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- 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**.

  mkorder –data "c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\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\outgoing\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.

  /opt/rational/clearcase/etc/mkorder –data /usr/tmp/to_goldengate –ship –copy –notify admin goldengate

  ```
  Shipping order
  "/var/adm/rational/clearcase/shipping/ms_ship/outgoing/sh_o_to_gold
  engate" 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 13, *Troubleshooting MultiSite Operations*

mkreplica

Creates a replica

Applicability

Product	Command type
MultiSite	multitool subcommand

Platform
UNIX
Windows

Synopsis

- Duplicate a VOB, generating a new replica object and 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* ...

- Import 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** | **–per·ms_preserve** [**–nprompt**] | **–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 **–tape** and **–ncaexported** options are valid only on UNIX.

Description

The creation of a new VOB replica is a three-phase process:

1. The **mkreplica –export** command duplicates the contents of the current VOB (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 255, 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. The packet is sent to one or more other sites.
3. At each receiving site, a **mkreplica –import** command uses the replica-creation packet to create a new VOB replica.

When a VOB is replicated for the first time, creating a second replica, the VOB's operation log (oplog) is enabled. All operations to be replicated are recorded in the oplog. Logging of operations continues until all but one of the replicas are deleted. Note that creation of additional replicas is recorded in oplog entries. Existing replicas learn about a new replica through the standard synchronization mechanism. (See the **syncreplica** reference page.)

Note: Before entering a **mkreplica –export** command, verify that 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).

Preservation Mode

When you enter a **mkreplica –import** command, you must choose the preservation mode. In any case, the user who enters the **mkreplica –import** command becomes the owner of the new replica. That user's group is the primary group of the VOB, and the user's group list becomes the VOB's group list. Preservation affects only element ownership and permissions. For more information about preservation modes, see *Identities and Permissions Strategy for VOB Replicas* on page 39.

Restrictions:

- You can create a replica that preserves identities and permissions only if its site supports the same user and group accounts as the originating site. On Windows, if all replicas in a family are not in the same Windows domain, the entire family cannot preserve identities and permissions. However, you can maintain identities and permissions preservation on the subset of replicas in the same domain.

  Note: There are no restrictions on creating permissions-preserving replicas.

- On Windows, the primary group of the user who enters the **mkreplica –import** command must be the same as the originating replica's group assignment.
- On UNIX, the user who enters the **mkreplica –import** command must belong to all the groups on the originating replica's group list.

To create a replica that preserves identities and permissions, you should run **mkreplica –export** at an identities- and permissions-preserving replica. To create a replica that preserves permissions, you should run **mkreplica –export** at an identities- and permissions-preserving replica or a permissions-preserving replica.

Note: After you create a new replica with **mkreplica –import –preserve** or **mkreplica –import –perms_preserve**, we recommend that you run **syncreplica –export** to inform other replicas in the VOB family about the preservation mode of the new replica.

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 includes one or more replica specifications, each of which indicates the new replica's name and the host on which a new replica is to be created.

The **–maxsize** option divides the single logical packet into multiple physical packets to conform to the limitations of the transfer medium.

Cleaning Up Used Packets

Replica-creation packets are not deleted after import. After you import a replica-creation packet with **mkreplica –import**, you must delete the packet.

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 run **mkreplica –export** on the host where the VOB storage directory resides.
- You can replicate a VOB replica to a host running an earlier major version of ClearCase only if the feature level of the exporting replica's VOB family is the same as the feature level of the ClearCase release on the target host. (You can always replicate a VOB to a host running a later major version of ClearCase.)

Options and Arguments — Export Phase

Specifying Temporary Workspace

Default

None.

–wor·kdir *temp-dir-pname*

Directory used 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 MultiSite Control Panel (Windows) or the file *ccase-home-dir*/config/services/shipping.conf (UNIX) 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*

The name of the first physical replica-creation packet. 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 method 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.

Handling Packet-Delivery Failures

Default

If a packet cannot be delivered, it is sent through the store-and-forward facility 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, the e-mail address of the administrator, and the notification program.

–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. This option overrides the expiration period specified for the storage class in the shipping.conf file (UNIX) or MultiSite Control Panel (Windows).

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-2002
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 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 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 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 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

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* on page 256.)

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

–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 a registry region for the new replica's VOB tag.

–opt·ions *mount-options*

Mount options for the new 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.

Preservation Mode

Default

None.

–pre·serve

Creates a replica that preserves identities and permissions. The user who enters the **mkreplica –import** command becomes the owner of the new VOB, and identities and permissions are preserved for all the elements in the new VOB.

–per·ms_preserve [**–nprompt**]

Creates a replica that preserves permissions. The user who enters the **mkreplica –import** command becomes the owner of the new VOB, and permissions are preserved for all the elements in the new VOB. The **–nprompt** option suppresses the prompt.

–npr·eserve

Creates a replica that is nonpreserving. 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. 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 /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
  /opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_1
  8-May-99.15:50:00_1
   - shipping order file is
  /opt/rational/clearcase/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
  /opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_1
  5-Aug-00.14.26.17_6011_1
  - shipping order file is
  /opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_repl_boston_
  hub_15-Aug-00.14.26.17_6011_1
  Dumping database...
  ...
  Dumper done.
  Attempting to forward/deliver generated packets...
  -- Forwarded/delivered packet
  /opt/rational/clearcase/shipping/ms_ship/outgoing/repl_boston_hub_1
  5-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 identities- and permissions-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 "identities and permissions preserving" –fship boston_hub bangalore buenosaires

  ```
  ...
  ```

- Similar to preceding example, but create the replica as permissions preserving.

  multitool mkreplica –import –workdir /tmp/ms_workdir –tag /vobs/dev –vob /net/goldengate/vobstg/dev.vbs –perms_preserve –c "create sanfran_hub replica" /tmp/sanfran_hub_packet

  ```
  multitool: Warning: In a permissions-preserving replica, cleartool
  protect operations will fail on client machines running ClearCase
  vesions associated with feature level 3 or lower.
  Should I create a permissions-preserving replica? [no] yes
  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...
  ...
  ```

multitool syncreplica –export –c "permissions preserving" –fship boston_hub bangalore buenosaires

```
...
```

- Similar to preceding example, but create the replica as a public VOB and nonpreserving. 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 13, *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 store-and-forward facility on each host. It provides controls for setting the configuration parameters described in the following sections. In some cases, the corresponding operation fails if a parameter is not defined, and in other cases there is a hard-coded default.

Maximum Packet Size

Default: 2097151 KB

Controls the splitting of 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.

Administrator E-mail

Default: None.

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. Verify that 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.
3. (optional) Enter a different value in the **Email Notification Program Path** field.

Email Notification Program Path

Default: *ccase-home-dir*\bin\notify.exe

Specifies the electronic mail program to be invoked in the circumstances listed in *Administrator E-mail*.

Timeout for Unreachable Host (minutes)

Default: None.

Specifies the number of minutes for the shipping server to wait before trying to contact a target host that was previously identified as unreachable.

If the shipping server tries to send a packet to a target host and determines that the host is unreachable, it creates a file in the *ccase-home-dir*\var\shipping\ms_downhost directory. The name of the file is the name of the unreachable host. If the **Timeout for Unreachable Host** setting is set, the shipping server checks the directory for target hosts during future shipping operations.

If the target host is found in the ms_downhost directory, and the difference between the current time and the last modification time of the file is less than the timeout setting on the shipping server host, the shipping server does not try to send packets to the target host. If the difference is equal to or greater than the timeout setting, the shipping server tries to send packets to the target host. If the **Timeout for Unreachable Host** setting is not set, the shipping server attempts to send the packet to the target host. (Each attempt to send a packet to an unreachable host takes about 30 seconds.)

Storage Classes

Storage Class Name

Default: 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.

Specifies the name of a storage class. For each storage class, you can specify values for packet expiration, the storage bay, the return bay, and the receipt handler.

Note: Storage class names are case sensitive.

Packet Expiration

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.

Specifies the expiration period (in days) for shipping orders associated with the specified storage class. This period begins at the time the shipping order is generated. If a packet cannot be delivered to all its 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 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 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*.

Storage Bay Path

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.

Defines the location of the 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 bay. Define ACLs on the storage 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, verify 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. For more information, see *MultiSite Installation* on page 31.

Note: When you create a new storage class, the storage bay and return bay that you specify are created. The incoming and outgoing directories in the bays are also created.

Return Bay Path

Default: When MultiSite is installed, 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.

Defines the location of the directory that holds the incoming and 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 on an NTFS file system inherit the Windows ACL on the bay. Define ACLs on the 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.

Receipt Handler Path

Default: None.

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 that value
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*	Name of the host 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 forwarded to its next destination.

Routing Information

The **Routing Information** settings control the network routing of packets.

Next Routing Hop

Default: None.

Specifies 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.

Destination Host Names

Default: None.

Packets destined for any host listed in this field are sent to the host specified in the **Next Routing Hop** box. You can specify a host using either its host name (which must be usable by hosts in different domains) or its numeric IP address. The value **–default** as the **Destination Hostname** accommodates all hosts that are not associated with a routing hop.

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 *Descriptions of Subcommands* on page 58.

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 these 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.

Commands that accept comment strings recognize one or more of the following 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
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. For more information about event records, see the **events_ccase** reference page in the *Command Reference*.

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*.

recoverpacket

Resets epoch number matrix so that 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 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 packet that is exported 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 epoch numbers at the sending replica 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 replicas 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 that are as old as the specified time, the **recoverpacket** command fails. In this case, the administrator at the receiving replica must use the **lsepoch** command to determine the correct epoch number, and the administrator at the sending replica must run **chepoch** on the sending replica to reset the epoch row. See the **chepoch** reference page and *Lost Update Packet* on page 191.

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 replica. 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-2002
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

Specifying the Row to Be Modified

Default

You must specify a replica. If you do not specify a *vob-selector*, the command uses the current VOB.

replica-selector

Updates 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)

Examples

- Reset the epoch row for replica **sanfran_hub** so that changes sent since last Monday are included in the next packet that is sent.

  multitool recoverpacket –nc –since Monday sanfran_hub

  ```
  Using epoch information from Monday 10/21/02 00:00:00
  Epoch row for replica "sanfran_hub" successfully reset.
  ```

- Reset the epoch row for replica **boston_hub** so that the changes included in the most recent update packet are included in the next packet that is sent.

  multitool recoverpacket –c "send latest packet" boston_hub@\dev

  ```
  Using epoch information from Thursday 10/24/02 14:55:34
  Epoch row for replica "boston_hub" successfully reset.
  ```

- 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 Saturday 10/19/02 05:05:45
  Epoch row for replica "bangalore" successfully reset.
  ```

multitool syncreplica –export –fship bangalore@/vobs/dev

```
Generating synchronization packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_2
2-Oct-02.15.44.28_4896_1
- shipping order file is
/opt/rational/clearcase/shipping/ms_ship/outgoing/sh_o_sync_boston_
hub_22-Oct-02.15.44.28_4896_1
Attempting to forward/deliver generated packets...
-- Forwarded/delivered packet
/opt/rational/clearcase/shipping/ms_ship/outgoing/sync_boston_hub_2
2-Oct-02.15.44.28_4896_1
```

See Also

chepoch, **lsepoch**, **restorereplica**
The Operation Log on page 22

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 11, *Implementing Requests for Mastership*.

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 or branch types, or all branches of a specific type. Even if replica-level access is enabled, the **reqmaster** command fails if requests are denied at the type level or object level. Use the **–deny** option to deny requests at the type and object level.

Note: Mastership request settings are not replicated. The **describe** command and the **Mastership** tab in the Properties Browser on Windows display the current replica's settings.

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`, reenter the command. If it continues to fail, ask the sibling replica's administrator 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 about error messages from the **reqmaster** command, see Chapter 11, *Implementing Requests for Mastership*.

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.

Other: You do not have to be logged on to a VOB server host to edit the mastership request ACL for a replica on that host. However, if you are not already on the ACL, both of the following conditions must be true in order for you to edit the ACL:

- You must be the VOB owner or privileged user.
- You must be logged on to a host in the same domain as the VOB server host.

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 (even if you are on the **–nusers** list): 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.
- Your host is running a later major version of ClearCase than the master replica's host.

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 (even if you are on the **–nusers** list): 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.
- Your host is running a later major version of ClearCase than the master replica's host.

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

Displays information about whether a request would succeed, but does not send a request for mastership.

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:main@/vobs/doc
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
  "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
  "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
  ```
  multitool: Error:  acc.c@@\main
  The following would block the "reqmaster" operation at replica
  "sydney".
  At least one checkout prevents the request.
  ```

- Request mastership of the branch **cmdsyn.m@@/main/v2.6_dev**.

  multitool reqmaster cmdsyn.m@@/main/v2.6_dev

  ```
  cmdsyn.m@@/main/v2.6_dev: Change of mastership at sibling replica
  "boston_hub" was successful.
  Mastership is in transit to the new master replica.
  ```

- Request mastership of the branch type **v2.0_integration**.

  multitool reqmaster brtype:v2.0_integration@vob:\tests

  ```
  brtype:v2.0_integration@vob:\tests: Change of mastership at sibling
  replica "sydney" was successful.
  Mastership is in transit to the new master replica.
  ```

See Also

chmaster

restorereplica

Replaces missing operations in a replica that has been restored 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

Warning: Execute this command immediately after you restore a replica from backup. Proceeding with normal development at a restored replica before executing this command causes irreparable inconsistencies among the replicas in a family.

restorereplica replaces missing changes in a VOB replica that has been restored from backup, as follows:

1. It causes the current replica to create special update packets that contain update requests to other replicas.
2. It locks the current replica's VOB object and marks the replica as being in the process of restoration.
3. It increments the recovery incarnation for the replica.
4. It causes **lsreplica –long** to indicate which replicas must send restoration updates to the current replica.

The current replica remains in the restoration state until you have received and applied (using **syncreplica –import**) all the restoration updates needed to bring the replica up to date with the state of the family. Collectively, these updates include all the changes to the family since the backup was made, including changes made in the current replica before its failure.

You cannot recover changes that were made after the last synchronization export from your current replica. For example, if your replica was backed up on Wednesday at 12:30 P.M. and your last synchronization export was Thursday at 3:00 P.M., you can recover all changes made until Thursday at 3:00 P.M. All changes made after that time are lost.

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 VOB Replicas* on page 198.

Locking the Replica

restorereplica locks the current replica's VOB object. Locking it ensures that while restoration proceeds through execution of **syncreplica –export** and **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 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.

Warning: If you use this optimization incorrectly, you 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 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 **syncreplica** command declares the replica to be restored completely only after updates are received from all other members of the VOB family and imported.

Warning: 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 family in an irreparably inconsistent state.

replica-selector ...

Specifies a subset of replicas from which updates are required before **syncreplica** declares the replica 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 VOB Replicas* on page 198.

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

Warning: To delete a replica, you must complete all steps described in *Deleting a Replica* on page 122, or synchronization and mastership problems can occur in other replicas in the family.

This command deletes from the current replica's database the VOB-replica object for another replica. Use this command to record the fact that another replica has been deleted.

Note: If running **rmreplica** makes the current replica the last member of the family, **rmreplica** turns off operation logging for this VOB and removes all oplogs, 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

For an example of removing a VOB-replica object, see *Deleting a Replica* on page 122.

See Also

chmaster, **mkreplica**

shipping.conf

Store-and-forward configuration file

Applicability

Product	Command type
MultiSite	MultiSite data structure

Platform
UNIX

Synopsis

/var/adm/rational/clearcase/config/shipping.conf

Description

This file controls the operation of the store-and-forward facility on each host. The file consists of comment lines (starting with #) and one or more configuration entries, and it 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/rational/clearcase/config/shipping.conf does not exist, the installation creates it by copying the template file. If /var/adm/rational/clearcase/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 MultiSite or the Rational Shipping Server in the default installation directory (/opt/rational/clearcase), you must edit the shipping.conf file and change /opt/rational/clearcase to the pathname of your installation directory.

Packet Size

MAX-DATA-SIZE *size* [**k** | **m** | **g**]

Default: 2097151 KB

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.

Notification

NOTIFICATION-PROGRAM *e-mail-program-pathname*

Default: /opt/rational/clearcase/bin/notify. This program is also used if no **NOTIFICATION-PROGRAM** entry exists.

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* . . .

Administrator Address

ADMINISTRATOR *e-mail-address*

Default: **root**

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.

Storage Bay and Return Bay

STORAGE-BAY *storage-class directory-pathname*
RETURN-BAY *storage-class directory-pathname*

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. This class is created when MultiSite is installed.

These lines define storage bay and return bay directories. A storage bay holds the outgoing and incoming update packets and shipping orders for a 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 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 bays does not matter.

Note: Storage class names are case sensitive.

MultiSite installation creates a default storage class named **–default**. The storage bay and return bay for this class are created on the local host in the /var/adm/rational/clearcase/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 in 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.

Expiration Period

EXPIRATION *storage-class number-of-days*
EXPIRATION –default *number-of-days*

Default: 14 days.

Specifies the expiration period (in days) for shipping orders associated with 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 destinations in the specified number of days, 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 value of **0** (zero) 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.

Packet Routing

ROUTE *next-hop host ...*
ROUTE *next-hop* **–default**

Default: None.

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.

Receipt Handler

RECEIPT-HANDLER *storage-class script-pathname*

Default: None.

Specifies a script for the shipping server to run for each packet received in a shipping bay. We recommend that you specify the **sync_receive** script as the **RECEIPT-HANDLER** entry. For example:

```
RECEIPT-HANDLER  –default
/opt/rational/clearcase/config/scheduler/tasks/sync_receive
```

shipping_server handles each packet that is received 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 that value.
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 option is used only when the packet is destined for this host.
–a·**ctual** *shipping-order-pname*	Location of the shipping order. This option is used only when the packet is destined for another host.
–s·**class** *storage-class*	Storage class associated with the packet. This option is used only if the packet was associated with a storage class when it was created.
–o·**rigin** *hostname*	Name of the host 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.

Port Numbers

CLEARCASE_MIN_PORT *port-number*
CLEARCASE_MAX_PORT *port-number*

Default: None.

Caution: Set these entries only on hosts that can communicate through the firewall and have been installed with the MultiSite shipping-server-only option. To use the shipping server on a firewall system, you must also set the CLEARCASE_MIN_PORT and CLEARCASE_MAX_PORT environment variables in the **clearcase** script. For more information, see *Specifying Port Values* on page 79.

These entries specify the range of ports for the 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.

Timeout Period for Unreachable Hosts

DOWNHOST-TIMEOUT *minutes*

Default: None.

Specifies the number of minutes for the shipping server to wait before trying to contact a target host that was previously identified as unreachable.

If the shipping server tries to send a packet to a target host and determines that the host is unreachable, it creates a file in the /var/adm/rational/clearcase/shipping/ms_downhost directory. The name of the file is the name of the unreachable host. If one of the following parameters is set, the shipping server checks the directory for target hosts during future shipping operations:

- **DOWNHOST-TIMEOUT** setting in the shipping.conf file
- SHP_DOWNHOST_TIMEOUT_RETRY environment variable

If both parameters are set, the shipping server uses **DOWNHOST-TIMEOUT**.

If the target host is found in the ms_downhost directory, and the difference between the current time and the last modification time of the file is less than the timeout setting on the shipping server host, the shipping server does not try to send packets to the target host. If the difference is equal to or greater than the timeout setting, the shipping server tries to send packets to the target host. If neither this setting nor the environment variable SHP_DOWNHOST_TIMEOUT_RETRY is set, the shipping server attempts to send the packet to the target host. (Each attempt to send a packet to an unreachable host takes about 30 seconds.)

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 and sends the associated packets or files to remote sites. After it delivers a file to all its destinations, **shipping_server** deletes the file unless one of the destinations is the local host.

Note: When **shipping_server** starts processing a shipping order, it locks the order. The lock 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 *Using Store-and-Forward Through a Firewall (UNIX only)* on page 76.

On UNIX, **shipping_server** forks one subprocess for each packet that it sends. As many as 10 **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 run 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 configuration settings in the shipping.conf file (UNIX) or MultiSite Control Panel (Windows). If a storage class is assigned multiple storage bays, available disk space determines the selection of a bay.

On 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.)

On 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 in file names.

Handling of File Name Conflicts

You can use the **mkorder** and **shipping_server** commands to transmit non packet files if the files are 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).

Setting a Timeout Period for Unreachable Hosts

You can set a timeout period during which the shipping server will not try to send packets to hosts that it previously identified as unreachable. For more information, see the **shipping.conf** (UNIX) or **MultiSite Control Panel** (Windows) reference page.

Log

On UNIX, **shipping_server** writes records of all packets sent and received, along with all errors, to file /var/adm/rational/clearcase/log/shipping_server_log.

On Windows, **shipping_server** writes records of all packets sent and received, notification messages, log messages, and all errors to the Windows Event Viewer. You can use the **cleartool getlog shipping** command to view **shipping_server** messages from the Event Viewer.

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

With **–poll**, processes all shipping orders in all outgoing storage bays and return bays on this host. With *sources*, processes all specified shipping orders.

–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 storage and return bays defined in the shipping.conf 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. However, 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

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- Process all shipping orders in all MultiSite storage bays.

  shipping_server –poll
  <*no output means command succeeded or did not find any shipping orders*>

- Process a particular shipping order. Note that the pathname argument specifies the shipping order file, not the data file to be transmitted.

  /opt/rational/clearcase/etc/shipping_server
  /var/adm/rational/clearcase/shipping/ms_ship/sh_o_sync_sydney_19-May-02.09:48:45_7660_1
  <*no output means command succeeded*>

- Process all shipping order files in a specified directory.

  shipping_server "c:\Program Files\Rational\ClearCase\var\shipping\ms_ship\outgoing"
  <*no output means command succeeded or did not find any shipping orders*>

- Process all shipping orders in the storage bays of a specified storage class.

  /opt/rational/clearcase/etc/shipping_server –poll –sclass daily
  <*no output means command succeeded or did not find any shipping orders*>

See Also

mkorder, **MultiSite Control Panel**, **shipping.conf**, **syncreplica**, **sync_export_list**
Chapter 13, *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 about 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 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 operations. If an administrator locks the VOB during operation 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/rational/clearcase/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 VOB locks.

Troubleshooting

sync_export_list fails if another **sync_export_list** process is exporting data from the same replica, unless export locking is disabled (see *Configuration File*). This failure prevents interference among export processes. 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, log files are stored in the /var/adm/rational/clearcase/log/sync_logs directory on UNIX and the *ccase-home-dir*\var\log directory on Windows. The file name includes the process ID of the **sync_export_list** command and the time (in UTC format) at which you ran the command.

The Weekly Log Scrubbing job installed with ClearCase deletes log files in /var/adm/rational/clearcase/log/sync_logs (UNIX) or *ccase-home-dir*\var\log (Windows) that have the prefix send or recv and the suffix _log and 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 about 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, and 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*

Tries a maximum of *num-tries* times to complete all exports successfully, and waits *num-seconds* seconds 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 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 that contains directives for **sync_export_list**. You must specify this argument last on the command line. You can include the following directives:

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** exports packets for replicas specified on the command line, and then 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

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- Send update packets from all replicas on the host to all their siblings.

  /opt/rational/clearcase/config/scheduler/tasks/sync_export_list –all

  ```
  SUCCESSFUL COMPLETION: log file removed.
  ```

- 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
```

- Generate update packets for replicas in the family **/vobs/dev**. Store the packets in the shipping bay, limit the packet size to 500 KB, and display messages during processing.

  /opt/rational/clearcase/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/rational/clearcase/log/sync_logs/send-000815-183301Z-6043
_log".
```

Files

UNIX

/var/adm/rational/clearcase/log/sync_logs
/var/adm/rational/clearcase/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 about 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 operations. If an administrator locks the VOB during operation 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/rational/clearcase/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 caused by 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 because of 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/rational/clearcase/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.

The Weekly Log Scrubbing job installed with ClearCase deletes log files in /var/adm/rational/clearcase/log/sync_logs (UNIX) or *ccase-home-dir*\var\log (Windows) that have the prefix send or recv and the suffix _log and 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 about 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/rational/clearcase/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, the following parameters are passed in automatically. You can use **–sclass**, **–data**, and **–actual** on 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

In these examples, the lines are broken for readability. You must enter each command on a single physical line.

- Import packets in the incoming storage bays for the **daily** storage class.

  /opt/rational/clearcase/config/scheduler/tasks/sync_receive –sclass daily

- Import a packet and apply it to all VOBs whose tags include the pattern **lib**.

**"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-02.18.17.54_6587_1"**

- On UNIX, specify **sync_receive** as the receipt handler for the **daily** storage class.

  **cp /opt/rational/clearcase/config/scheduler/tasks/sync_receive*
  /var/adm/rational/clearcase/scheduler/tasks**

  Edit the shipping.conf file and add a receipt handler entry:

  **RECEIPT-HANDLER daily
  /var/adm/rational/clearcase/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/rational/clearcase/log/sync_logs
/var/adm/rational/clearcase/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

Exports or imports 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]
  {
  { **–shi·p** | **–fsh·ip** } [**–scl·ass** *storage-class*] [**–pex·pire** *date*] [**–not·ify**
  e-mail-addr]
  | **–tap·e** *raw-device-pname*
  | **–out** *packet-file-pname*
  }
  replica-selector ...

- 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*]
  | **–tap·e** *raw-device-pname*
  | { *packet-file-pname* | *staging-area-pname* } ...
  }

Note: The **–tape** option is valid only on UNIX.

Description

Synchronization of an existing replica with one or more sibling replicas is a three-phase process:

1. At one site, a **syncreplica –export** command creates an update packet that contains changes that have occurred in the replica at that site (and perhaps other replicas, as well).
2. The packet is sent to one or more other sites.
3. At another site, a **syncreplica –import** command applies the changes in the update packet to its replica of the same VOB.

Step 3 occurs at all sites that receive the packet.

Contents of an update packet:

- All changes that have occurred in the current 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 those particular replicas only.

Notes on the Export Phase

MultiSite is designed for efficient updating of replicas. **syncreplica –export** attempts to exclude 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 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 327.

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 327.

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 imported 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 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.

For more information about update failures, see Chapter 13, *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.

Preservation of Identities and Permissions

If a VOB replica preserves identities and permissions, **syncreplica –import** maintains the consistency of identities and permissions information for elements mastered by the VOB family's identities- and permissions-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 preserves permissions only, **syncreplica –import** maintains the consistency of permissions information for elements mastered by the VOB family's identities- and permissions-preserving replicas and permissions-preserving replicas. Changes to identities for 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 VOB. (This is true even if the **root** user or a member of the ClearCase administrators group imports the packet.)

If a VOB replica is nonpreserving, changes to identities 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 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. Identities and permissions changes made at nonpreserving 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 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: syncreplica waits only if it detects the lock before it starts processing operations. If an administrator locks the VOB during 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.

Other: You must run **syncreplica** on the host where the VOB storage directory resides.

Options and Arguments — Export Phase

Specifying the Update Packet Size

Default

If you do not specify **–maxsize**, the default packet size depends on the shipping option:

- 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.

If you use **–ship** or **–fship** and omit the **–sclass** 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/rational/clearcase/shipping/ms_ship on UNIX and *ccase-home-dir*\var\shipping\ms_ship on Windows.

–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.

–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 replica until the first packet is received and processed.

–out *packet-file-pname*

The name of the first update packet. 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 method 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 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, the e-mail address of the administrator, and the notification program.

–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. This option overrides the expiration period specified for the storage class in the shipping.conf file (UNIX) or MultiSite Control Panel (Windows).

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-2002
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 ...

Specifies the replicas to which you want to send update packets. These replicas must be in the same VOB family. Specify *replica-selector* in the form [**replica:**]*target-replica-name*[@*source-vob-selector*]

target-replica-name	Name of the replica to which you want to send the packet (you can display replica names with **lsreplica**)
source-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

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 the current host's storage bays. Any unprocessed update packets intended for this host are applied to the appropriate replicas on the host. With **–sclass**, **syncreplica** scans only 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-02.09.33.02_3447_1
   - shipping order file is c:\Program
  Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\sh_o_sync_ba
  ngalore_19-May-02.09.33.02_3447_1
  ```

- Similar to preceding example, but ship the packet immediately.

  multitool syncreplica –export –fship boston_hub@\dev

```
Generating synchronization packet c:\Program
Files\Rational\ClearCase\var
\shipping\ms_ship\outgoing\sync_bangalore_19-May-02.09.33.02_3447_1
 - shipping order file is c:\Program
Files\Rational\ClearCase\var\shipping\ms_ship\outgoing\sh_o_sync_ba
ngalore_19-May-02.09.33.02_3447_1
Attempting to forward/deliver generated packets...
 -- Forwarded/delivered packet c:\Program
Files\Rational\ClearCase\var
\shipping\ms_ship\sync_bangalore_19-May-02.09.33.02_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-02.09.45.01_7634_
1
to VOB \\ramohalli\vobs\dev.vbs
```

See Also

mkorder, **mkreplica**, **MultiSite Control Panel**, **shipping.conf**, **sync_export_list**
Chapter 13, *Troubleshooting MultiSite Operations*

Index

D

E

F

H

I

L

M

N

O

P

R

S

T

U

V