Rational Software Corporation®

Rational ClearQuest MultiSite®

Administrator’s Guide

VERSION: 2002.05.00

PART NUMBER: 800-025263-000

WINDOWS/UNIX

Rational® the software development company

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

Contents

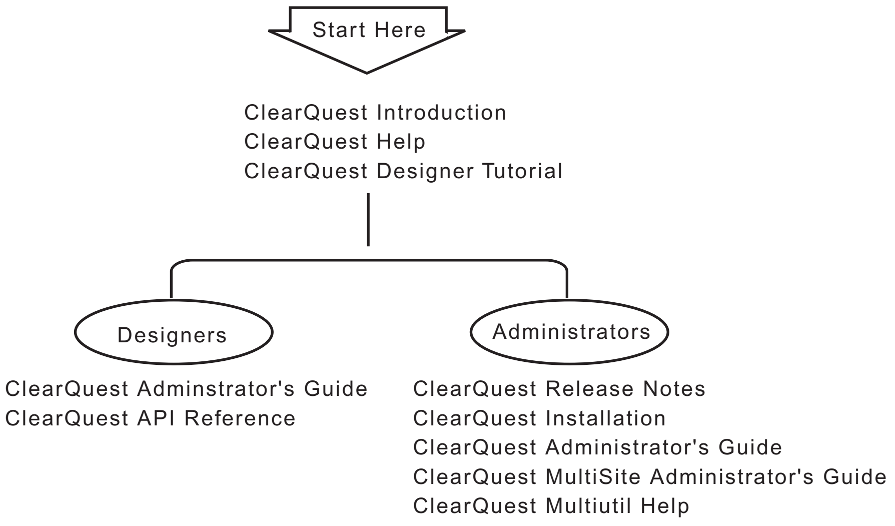
Preface

ClearQuest® MultiSite® is a layered product option to Rational ClearQuest. It supports parallel software development across project teams distributed geographically and provides automated replication of databases and transparent access to all database records.

About this manual

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

ClearQuest documentation roadmap


Start Here
ClearQuest Introduction
ClearQuest Help
ClearQuest Designer Tutorial
Designers
Administrators
ClearQuest Adminstrator's Guide
ClearQuest API Reference
ClearQuest Release Notes
ClearQuest Installation
ClearQuest Administrator's Guide
ClearQuest MultiSite Administrator's Guide
ClearQuest Multiutil Help

Typographical conventions

This manual uses the following typographical conventions:

- *clearquest-home-dir* represents the directory into which the ClearQuest Product Family has been installed. By default, this directory is /usr/atria on UNIX and C:\Program Files\Rational\ClearQuest on Windows.
- **Bold** is used for names the user can enter; for example, all command names, file names, and branch names.
- *Italic* is used for variables, document titles, glossary terms, and emphasis.
- `A monospaced font` is used for examples. Where user input needs to be distinguished from program output, **bold** is used for user input.
- [ ] 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.
- If a command or option name has a short form, a "medial dot" (·) character indicates the shortest legal abbreviation. For example:

  sc·lass

  This means that you can truncate the command name to **sc** or any of its intermediate spellings (**scla** and so on).

Online documentation

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

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

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

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

Contacting Rational Technical Publications

To send feedback about documentation for Rational products, please send e-mail to our technical publications department at techpubs@rational.com.

Contacting Rational Software

If you have a technical problem and you cannot find the solution in this guide or in the online Help, contact Rational Software Technical Support.

Location	Telephone	Facsimile	E-mail
North America	(800) 433-5444 (toll free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0) 20-4546-200 Netherlands	+31 (0) 20-4545-201 Netherlands	support@europe.rational.com
Asia Pacific	+61-2-9419-0111 Australia	+61-2-9419-0123 Australia	support@apac.rational.com

Note: When you contact Rational Technical Support, please be prepared to supply the following information:

- Your name, telephone number, and company name
- Your computer's make and model
- Your operating system and version number
- Product release number and serial number
- Your case ID number (if you are following up on a previously-reported problem)

Introducing ClearQuest MultiSite 1

ClearQuest MultiSite® adds a powerful capability to Rational ClearQuest. With ClearQuest MultiSite, developers at different locations can use the same schema and user database. Each location (*site*) has synchronized copies (*replicas*) of the user database and its respective schema repository. At any time, a site can propagate the changes made in its particular replicas to other sites, by sending update packets. The update process can be scheduled by any number of methods to be automatic or can be started manually.

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

Understanding the architecture of ClearQuest MultiSite

Before you start using ClearQuest MultiSite, you should be familiar with the architecture of ClearQuest.

ClearQuest MultiSite uses replicas to distribute databases among several sites. Remember that within ClearQuest, a user database cannot exist without its corresponding schema repository; therefore ClearQuest MultiSite replicates both a schema repository and an associated user database when it creates a replica site.

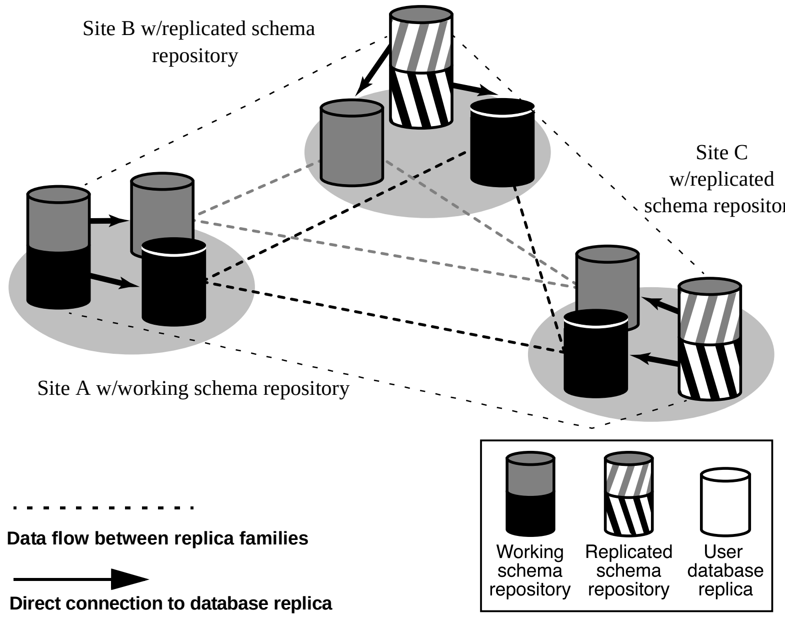
Consequently, when you work with ClearQuest replicas, you are working with two physical databases, a replicated schema repository and a user database replica.

ClearQuest MultiSite introduces the following new terms:

Term	Definition
Clan	A clan represents a group of replicas that use the same schema repository (a database set). A clan consists of two or more replica families that reside at multiple *sites*. Replicas originating from the same database set use the same clan name. The clan is named when the first replica is activated.
Site	A named collection of replicas at the same physical location. More than one clan can reside at a site. Within a clan, there are two types of sites, a working schema repository site and a replicated schema repository site.
Family	Within a clan, replicas are grouped into replica families. A replica family consists of all the replicas of a specific database. Schema repositories and user databases both participate in families. The family name is the same name as the five-character logical database name of the originating ClearQuest database.
Replica	A copy of a ClearQuest database or its associated schema repository. To refer to a replica, use the site name and family name.
Host	The LAN name or IP address of the network node that has access to a site. The host name is used with the Rational Shipping Server and refers to the machine where the storage bays for the respective replica are located and must be consistent with the name of the host specified when you configured the MultiSite Control Panel.

Using clans

A clan represents all the replicas within a database set (a collection of databases which includes a single schema repository and its associated databases).


Site B w/replicated schema repository
Site C w/replicated schema repository
Site A w/working schema repository
Data flow between replica families
Direct connection to database replica
Working schema repository
Replicated schema repository
User database replica

Understanding the role of the working schema repository

Within a ClearQuest MultiSite clan, there are two types of sites: a working schema repository site and a replicated schema repository site. Each site can perform different tasks and has a different role within MultiSite.

The working repository site within a clan is used to propagate changes to schemas and to create additional user databases which can then be replicated. Only the working schema repository can make schema changes and a clan can have only one working schema repository.

Type of site	Supported tasks	Unsupported tasks
replicated schema repository site	• Runs multiutil commands from administrator's workstation • Installs and configures Store-and-Forward (Rational Shipping Server), if necessary • Can submit new records • Modifies records of which it has mastership • Administers users of which it has mastership	Within the MultiSite clan, • Cannot modify schemas • Cannot create new ClearQuest databases
working schema repository site	• Runs multiutil commands from administrator's workstation • Installs and configures Store-and-Forward (Rational Shipping Server), if necessary • Can submit new records • Modifies records of which it has mastership • Administers users of which it has mastership • Modifies schemas • Creates new ClearQuest databases	None

Using families

Within a clan, replicas are grouped into replica families. A replica family consists of all the replicas of a specific database.

Both schema repositories and user databases participate in replica families. The family name is the same as the ClearQuest database name. The schema repository family name is always MASTR.

There are at least two database families represented at each replica site. One for the user database replica and one for the replicated schema repository.

Updates for a site can contain updates to either the user database replica, the replicated schema repository or both.

Using sites

A site is a named collection of replicas in the same clan that reside at the same location. Each site has a replicated schema repository and at most one replica from each user database family. Each site is served by a host machine, which is responsible for receiving and sending update packets to replicas within its family.

Understanding mastership

In a ClearQuest MultiSite environment, tracking changes and preventing data corruption is accomplished with an exclusive-right-to-modify, called *mastership*. Mastership determines when a user of a database replica is allowed to modify data. For example, without using mastership, users could modify records in different replicas independently and simultaneously, the result would be chaos. For example, if a record SAMPL00001 was modified in three replicas at the same time it would be impossible to determine which was the real record SAMPL00001, and what ought to happen to the other two versions.

With mastership, database records and other ClearQuest objects such as users and queries are assigned a *mastering replica*. The initial mastering replica of a ClearQuest object is the site where the object is created. The mastering replica can be changed subsequently (see Chapter 6, *Managing mastership*).

In general, an object can be modified or deleted only at its mastering replica. With database records, mastership information is stored as a field value in a record. To change the mastership of a record (and to allow modifying) users can change the value of the mastership field to the current site. Mastership of the record is then changed during the next synchronization cycle. All other ClearQuest objects require an administrator to change the mastership.

The following ClearQuest objects have a mastering replica:

- Records
- Users and groups
- Workspace items (queries, reports, charts and folders)
- Schema repository

Adjusting your workflow for use with ClearQuest MultiSite

When working in a ClearQuest MultiSite environment, you'll need to adjust your workflow to account for any stage in your software lifecycle that requires a change of mastership for a record or defect.

For example, users at a site in Paris, France can submit defects that should be worked on by developers in Boston, USA. But without a change of mastership, developers in Boston will not be able to modify defects entered by users in Bangalore.

You can modify your ClearQuest schema to automate such changes by using field and action hooks, see *Using mastership with records* on page 72.

Synchronizing replicas in a family

Because the data in a replicated database is modified by multiple sites, the contents of each database replica in a *family* tend to diverge. In fact, a particular replica is rarely—and need never be—in the same state as any other replica. To keep the replicas from diverging too much, each site sends updates to one or more other sites. Updating a replica may change both its database and its schema repository to reflect the activity that has taken place in one or more other replicas.

Replica information is exported from a database in packets. A *logical packet* includes all the information required to create a new database replica (replica-creation packet) or to update one or more existing database 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* to be copied to disk.

After a logical packet is created and sent to a site, it is processed at that site. The changes that occurred originally at the sending site (and perhaps some other sites, too) are added to the database and schema repository (if needed) of the replica at the receiving site. If the logical packet includes several physical packets, the import commands always process the physical packets in the correct order. No error occurs if the same packet is imported two or more times at a site.

You can match the synchronization strategy for each database family to its particular use patterns, your organization's needs, and the level of connectivity among the sites. For one database, you can update replicas every hour, using a high-speed network; for another database, you can send updates only once or twice a month, using electronic mail, magnetic tape or disk files as the delivery mechanism. See *Planning a replication strategy* on page 26 for information on planning synchronization. Chapter 4, *Synchronizing replicas*, discusses how to create and synchronize database replicas. *Database operations and the oplog on page 8* describes the underlying mechanism that supports synchronization and replication.

ClearQuest MultiSite, time, and time zones

In ClearQuest 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 modifies a record at 14:33 Bangalore time, the modification time is stored as 09:03UTC. When a developer in San Francisco looks at the description of the version, the modification 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 *Planning a replication strategy* on page 26.

Conflict resolution

Mastership restrictions prevent most inconsistent changes in different database replicas, but some inconsistent changes are unavoidable, particularly in the naming of new users, groups and other stateless record types. For example, a new user named *jsmith* can be created at two or more replicas during the same time period between sychronizations.

To avoid such naming conflicts, the ClearQuest MultiSite administrators for a family need to create and enforce some naming conventions for database records. A ClearQuest use model that is used consistently across sites reduces the potential for conflicts. For example, the administrators for a family can agree that all site-specific records (such as user names) must include a site identifier, or that stateless record types such as user and e-mail rules that will be used at multiple sites are created only at a certain site, which forces the name to be unique.

When naming conflicts are found, ClearQuest MultiSite internally renames the conflicting records. If this happens, you should rename the conflicting records as soon as possible, see *Resolving naming conflicts* on page 95.

Database operations and the oplog

This section describes the database replica mechanism that supports replica synchronization. This information is not required to use ClearQuest MultiSite, but is helpful when you want to deepen your understanding of the error-recovery facilities. See Appendix A, *Troubleshooting MultiSite operations*.

For a replicated database, operation log entries (*oplogs*) are created. These entries store all the information required to replay the changes in another replica:

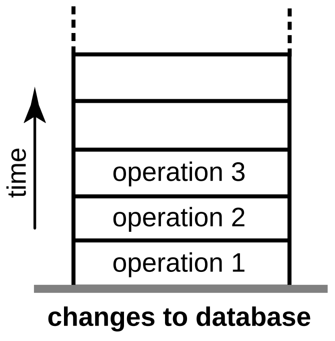
- The identity of the replica where the change originally took place.
- The specific changes to a database record or to a schema in the schema repository made during a single checkout; for example, submission of a new record, schema updates, and so on.
- An integer sequence number: 1 for the first change originating at a particular replica, 2 for the next change, and so on. This is called the *epoch number* of the oplog entry.

The exact kind and amount of information varies with the specific operation. For example, an oplog entry for the submission of a new record has different, and more, information than an oplog entry for modifying an existing record.

Note: Oplog entries are created only for replicated databases. You can delete a replica's oplog entries after they have been used to update other replicas. For more information, see *Scrubbing parameters for database replicas* on page 67.

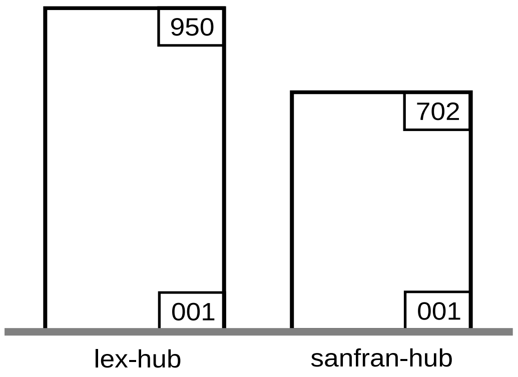
Tracking operations for each replica

The history of an unreplicated database is a linear sequence of operations, as shown.



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

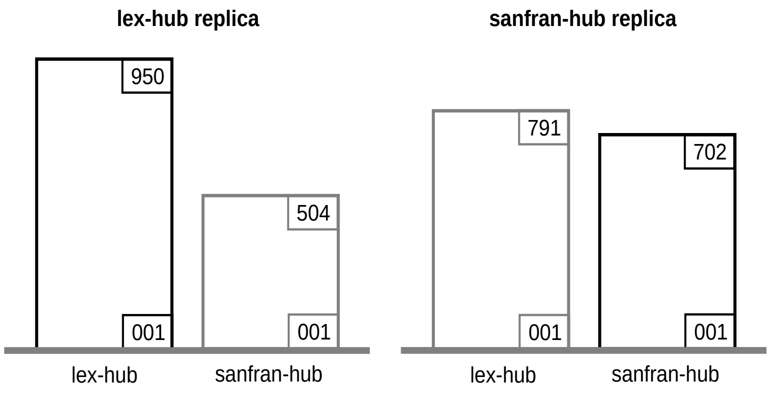
The figure below shows the state of two replicas in a family:



Note: Operations with epoch numbers 1-950 have occurred at replica **lex_hub** and operations 1-702 have occurred at replica **sanfran_hub**.

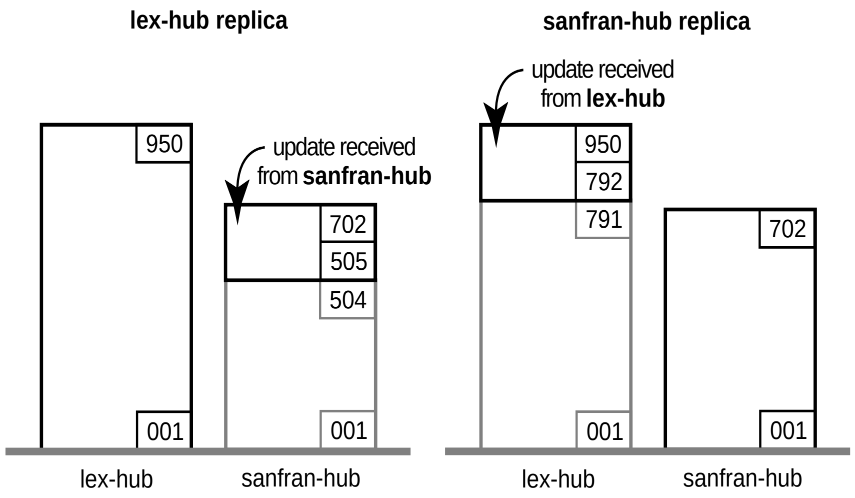
A replica has accurate data only about its own operations. Until it receives update packets, its information about other replicas is out of date. For example, replica **lex_hub** records 950 local operations, but has received update packets for only 504 **sanfran_hub** operations. Similarly, replica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

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



Epoch numbers

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



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

Optimization and the epoch number matrix

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

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

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

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

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 as follows:

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

Note: A a **syncreplica -export** command updates epoch numbers immediately. It does not wait for acknowledgment from the receiving site that the packet has been received and applied correctly.

When the **lex_hub** replica receives an update from **sanfran_hub**, it revises its own row (**lex_hub**) and the **sanfran_hub** row in its epoch number matrix.

	Operations originated at **lex-hub**	Operations originated at **sanfran-hub**
lex-hub's record of its own state	950	504
lex-hub's estimate of **sanfran-hub**'s state	912	504

During normal ClearQuest and ClearQuest 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 94.

The contents of this matrix are reported by the **multiutil lsepoch** command at the **lex_hub** replica:

```
multiutil lsepoch -clan telecomm -site lex_hub -family PRODA
-user lexadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at
site 'lex_hub' (@host1):
lex_hub: 950
sanfran_hub: 504
Multiutil: Estimates of the epochs from each site replayed at
site 'sanfran_hub' (@host2):
lex_hub: 912
sanfran_hub: 504
```

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

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

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

   ```
   multiutil lsepoch -clan telecomm -site lex_hub -family PRODA
   -user lexadmin -password secret
   Multiutil: Estimates of the epochs from each site replayed at
   site 'lex_hub' (@host1):
   lex_hub: 950
   sanfran_hub: 504
   Multiutil: Estimates of the epochs from each site replayed at
   site 'sanfran_hub' (@host2):
   lex_hub: 950
   sanfran_hub: 504
   ```

Indirect synchronization

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

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

The figure below shows replica **lex_hub's** epoch number matrix.

	Operations originated at **lex-hub**	Operations originated at **sanfran-hub**	Operations originated at **bangalore**
lex-hub's record of its own state	950	504	653
lex-hub's record of **sanfran-hub**'s state	912	504	653
lex-hub's record of **bangalore**'s state	709	221	653

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

```
multiutil lsepoch -clan telecomm -site lex_hub -family PRODA
-user lexadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at
site 'lex_hub' (@host1):
lex_hub: 950
sanfran_hub: 504
bangalore: 653
Multiutil: Estimates of the epochs from each site replayed at
site 'sanfran_hub' (@host2):
lex_hub: 912
sanfran_hub: 504
bangalore: 653
Multiutil: Estimates of the epochs from each site replayed at
site 'bangalore' (@host3):
lex_hub: 709
sanfran_hub: 221
bangalore: 653
```

A **syncreplica -export** command at Lexington uses this matrix to export an update for **bangalore**:

1. At Lexington, there are 950 local operations (number in upper-left corner of matrix), and the estimate is that **bangalore** has been updated only to epoch number 709 (lower-left corner).
2. For operations that originated at **sanfran_hub**, **lex_hub** has been updated to epoch number 504, and estimates that **bangalore** has been updated only to epoch number 221.
3. The update packet that **lex_hub** sends to **bangalore** includes **lex_hub** oplogs 710-950 and **sanfran_hub** oplogs 222-504. The output of a **multiutil lsepoch** command at Lexington now looks like this:

```
multiutil lsepoch -clan telecomm -site lex_hub -family PRODA
-user lexadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at
site 'lex_hub' (@host1):
lex_hub: 950
sanfran_hub: 504
bangalore: 653
Multiutil: Estimates of the epochs from each site replayed at
site 'sanfran_hub' (@host2):
lex_hub: 912
sanfran_hub: 504
bangalore: 653
Multiutil: Estimates of the epochs from each site replayed at
site 'bangalore' (@host3):
lex_hub: 950
sanfran_hub: 504
bangalore: 653
```

Example: Synchronization and the epoch number matrix

The following example illustrates how the epoch number matrix changes at various sites as replica creation and synchronization occur.

1 After activation, but before replication is enabled for the first time at lex_hub, its epoch number matrix is empty:

```
multiutil lsepoch -clan telecomm -site lex_hub -family PRODA
-user lexadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at
each site ‘LEX_HUB’ (@host1):
LEX_HUB: 0
```

2 After the administrator at the sanfran_hub site imports the replica-creation packet, the epoch number matrix for sanfran_hub looks like this:

```
multiutil lsepoch -clan telecomm -site sanfran_hub -family
PRODA -user sfadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at
each site ‘SANFRAN_HUB’ (@host2):
LEX_HUB: 2
SANFRAN_HUB:0
```

Note: The important thing to note is that sanfran_hub’s estimate of itself is 0. In this example, sanfran_hub also estimates that lex_hub committed two database operations. These operations could range from a new record to the creation of a replica.

3 As development work occurs at the two replicas, each replica’s record of its own state is updated accordingly. However, because no synchronization has taken place, each replica’s estimate of the other replica’s state does not change.

At lex_hub:

```
multiutil lsepoch -clan telecomm -site lex_hub -user lexadmin
-password secret lex_hub
Multiutil: Estimates of the epochs from each site replayed at
each site ‘LEX_HUB’ (@host1):
LEX_HUB: 12
SANFRAN_HUB:0
```

At sanfran_hub:

```
multiutil lsepoch -clan telecomm -site sanfran_hub -user
sfadmin -password secret
Multiutil: Estimates of the epochs from each site replayed at
each site ‘SANFRAN_HUB’ (@host2):
LEX_HUB: 2
SANFRAN_HUB:7
```

4 The administrator at `lex_hub` enters a **syncreplica** `–export` command to generate an update packet for `sanfran_hub`.

5 The `sanfran_hub` row is updated to show that all operations that have taken place at `lex_hub` will be applied at the `sanfran_hub` replica:

```
multiutil lsepoch -clan telecomm -site lex_hub -user lexadmin
-password secret sanfran_hub
Multiutil: Estimates of the epochs from each site replayed at
each site ‘SANFRAN_HUB’ (@host1):
LEX_HUB: 12
SANFRAN_HUB:0
```

6 At `sanfran_hub`, the administrator applies the update packet. The epoch number matrix for `sanfran_hub` now reflects the changes made at the `lex_hub` replica:

```
multiutil lsepoch -clan telecomm -site sanfran_hub -user
sfadmin -password secret sanfran_hub
Multiutil: Estimates of the epochs from each site replayed at
each site ‘SANFRAN_HUB’ (@host1):
LEX_HUB: 12
SANFRAN_HUB:7
```

Using the multiutil commands

This section summarizes the commands available with ClearQuest MultiSite and the ClearQuest API functions that display or modify ClearQuest MultiSite information. Reference pages for the ClearQuest MultiSite commands are available in Appendix C, *MultiSite Command Line Reference*, and are also available online:

- On Windows, the ClearQuest MultiSite **multiutil man** command displays reference pages in Windows Help.
- On UNIX, the ClearQuest MultiSite **multiutil man** command displays usage information.
- On both platforms, the Administering ClearQuest MultiSite manual, which includes the reference pages, is available in PDF format in the following directories: On Windows, \\ClearQuest\books. On UNIX, \\ClearQuest books.

Using multiutil

multiutil has a set of subcommands that perform product functions, such as replica creation, synchronization, and management; change of mastership of objects stored in replicas; and failure recovery.

- Command options can sometimes be abbreviated to three characters and sometimes fewer, as indicated in the reference pages.
- Command options must always be typed in lowercase.
- You can use **multiutil** in *single-command mode*. For example:

  ```
  multiutil lsreplica -clan telecomm -site boston -user
  bostonadmin -password secret -short
  ```

  Also in *interactive mode*:

  ```
  c:\multiutil
  multiutil> lsreplica -clan telecomm -site boston -user
  bostonadmin -password secret -short
  multiutil> quit
  ```

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

  ```
  multiutil help lspacket
  Usage: lspacket [-long | -short] [pathname ...]
  multiutil man lspacket
  ```

Running multiutil commands at multiple machines at the same site

By default, only one machine per site is configured to administer ClearQuest MultiSite and use the multiutil commands. This machine is designated in two ways:

- By running **multituil activate**. The machine where multiutil activate is run is automatically configured to run subsequent multiutil commands.
- By running **mkreplica -import**. The machine where multiutil mkreplica -import is run is automatically configured to run subsequent multiutil commands.

Configuring additional machines to use multiutil

If you want to run multiutil from a machine other than the one used to run **activate** or **mkreplica -import**, you'll need to configure the machine to access the replicated schema repository (database set) at your site.

The name of the replicated schema repository at your site (database set) is always in the following format: CQMS.<CLAN_NAME>.<SITE_NAME>

On UNIX, you'll need to use the ClearQuest's **cqreg add_dbset** subcommand. For more information about this command, type **man cqreg** at a UNIX prompt.

On Windows, you can do this with ClearQuest's **installutil adddbset** command.

The syntax for the command is:

> *<your ClearQuest path>* \ **installutil adddbset** *<dbset_name> <db_vendor> <server_hostname> <db-path \ filename.suffix or database name> <ro-login name> <ro-login password> <connection options>*: for connection options, use "" if not Oracle; if Oracle, HOST=<host>;SID=<sid>; server_ver=<ver7>; client_ver=<ver>

This command is used to allow a ClearQuest client to access multiple schema repositories. In the case of ClearQuest Multisite, the multiutil command is functioning as a client which needs access to the replicated schema repository (database set) at your site.

The name of the replicated schema repository at your site (database set) is always in the following format: CQMS.<CLAN_NAME>.<SITE_NAME>

The following example illustrates how to use the **installutil adddbset** command at the command prompt to connect to an the replicated schema repository for the *telecomm* clan at the *boston* site. Notice that the <dbset_name> used is CQMS.TELECOMM.BOSTON.

```
E:\Program Files\Rational\ClearQuest>installutil adddbset
CQMS.TELECOMM.BOSTON ORACLE bar_host cquser cquser password
client_ver=7.0
```

For more information about installutil and connecting to schema repositories, see the *Administering Rational ClearQuest* manual.

Multiutil subcommands

The following tables describe the different kinds of **multiutil** subcommands.

Utility, replica creation, synchronization and management

The majority of multiutil commands are used for replica creation and synchronization and are included in the following table.

Command	Platform	Description
cd		Changes current working directory
help	Windows/UNIX	Displays **multiutil** command syntax
man	Windows/UNIX	Displays a ClearQuest MultiSite reference page on Windows. On UNIX, displays command syntax.
exit	Windows/UNIX	Ends interactive **multiutil** session
quit	Windows/UNIX	Ends interactive **multiutil** session
activate	Windows/UNIX	Prepares a database set to be used by ClearQuest MultiSite.
chreplica	Windows/UNIX	Changes the properties of a replica.
dumpoplog	Windows/UNIX	Displays the content of a replica’s oplog.
lspacket	Windows/UNIX	Describes contents of packet.
mkreplica	Windows/UNIX	Creates a database replica.
rmreplica	Windows/UNIX	Deletes a replica.
scruboplog	Windows/UNIX	Deletes oplog entries.
syncreplica	Windows/UNIX	Generates or applies update packets.

Object mastership

To prevent conflicting changes from occurring at different replicas of a database, certain database objects are assigned a *mastering replica*. The initial master of an object is the replica where the object is created. For more information on mastership, see *Managing mastership* on page 69.

Command	Description
chmaster	Transfers mastership of ClearQuest objects including records, users, groups and Workspace items.
describe	Describes a replica

Failure recovery

Each database uses an *epoch number matrix* to track its own level *and* the epoch level of all other replicas. (Because replicas are always changing, a replica knows what changes have been made to itself; but it can have only an estimate of the states of other replicas.) Each time a replica sends an update packet, it updates its own epoch number matrix, under the assumption that the packet will be delivered to its destinations and applied to the appropriate replicas. For more information, see *Database operations and the oplog on page 8*.

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

Command	Description
chepoch	Changes epoch numbers.
lsepoch	Lists the current epoch estimates for specified replicas.
recoverpacket	Resets epoch row table so changes in 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.

Additional MultiSite commands

The ClearQuest MultiSite commands that are not build into multiutil are listed below:

Command	ClearQuest-home-dir Location	Description
mkorder	**etc** (UNIX) **bin** (Windows)	Creates a shipping order for use by *store-and-forward*.
shipping_server	**etc** (UNIX) **bin** (Windows)	*Store-and-forward* packet transport server.

Specifying replicas in commands

When you specify replicas in a multiutil command, you'll need to indicate site, family and clan, when necessary. If there is only one clan at your site, the `-clan` argument is optional. The `-site` argument is also optional, except when creating replicas.

For example, the following command logs into the PRODA replica family at the Boston site which is a member of the telecomm clan. Notice that you must also supply the ClearQuest user login name and password.

```
multiutil lsreplica -clan telecomm -site boston -family PRODA
-user bostonadmin -password secret
```

ClearQuest MultiSite API functions

You can use the ClearQuest API in hooks and external applications to determine whether or not you are working with a replicated database and if you have mastership of the ClearQuest record or object you want to modify.

Here is a list of three of the more useful ClearQuest API methods associated with ClearQuest MultiSite, for a complete list of ClearQuest MultiSite API commands, see the *ClearQuest API Reference* manual.

API method	Associated object	What it does
SiteHasMastership	Entity, Workspace, User	Returns the value that indicates which site currently has mastership of a record, Workspace item, user or group
GetSiteExtendedName	Workspace, User, Entity	Returns the value of the ratl_keysite name which allows to determine which records, users or groups have naming conflicts and need to be renamed
GetLocalReplica	Session object	Allows you to determine if the database you are working with is a replica. Also can list replica information.

Deploying ClearQuest MultiSite

2

Deploying ClearQuest MultiSite requires setup tasks additional to installing Rational ClearQuest. You'll need to make decisions about how to transport update packets, what your sychronization strategy will be, and modify your schema to use the ClearQuest MultiSite mastership field. Deploying ClearQuest MultiSite involves the following tasks:

- Before you begin
- Planning a replication strategy
- Defining mastership policies
- Defining a transport mechanism
- Overview of deployment tasks

Before you begin

Before you begin to use ClearQuest MultiSite, you should review this chapter. Implementing ClearQuest MultiSite includes planning a replication strategy and may also include ClearQuest schema changes that will enable you to change your defect-tracking process for use with ClearQuest MultiSite.

1. Install ClearQuest MultiSite and configure licensing. For more information, see *Installing Rational ClearQuest* and *Administering Licenses for Rational Software*.
2. Plan your replication strategy, including pattern and schedule.
3. Define your mastership policies.
4. Choose and configure your transport mechanism.
5. Deploy ClearQuest MultiSite in a test scenario before allowing users to use a replicated database. For a detailed example of a test deployment scenarios, see *Overview of deployment tasks* on page 33.

Planning a replication strategy

When you plan your replication strategy, you think about what type of sychronization pattern you want to use and what schedule you want to follow.

As you plan your sychronization pattern, remember that each site's update packet includes both its own changes and changes it receives from other sites. With the right strategy, there can be no need for each site to send update packets to every other site within the family.

Rational recommends that you document your plan and implement your design decisions in a set of test replicas before implementing your ClearQuest MultiSite environment.

Synchronization patterns

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

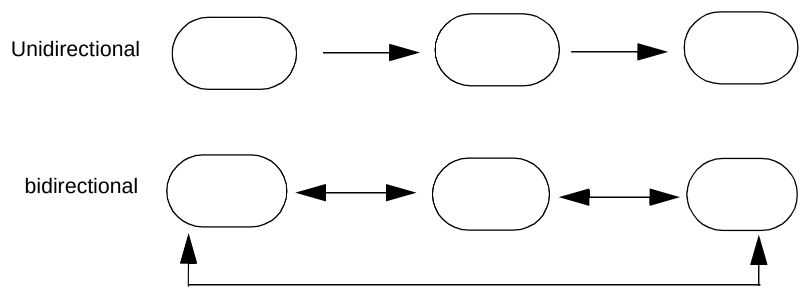
- Bandwidth between sites
- Network topology

- Timing: how quickly do changes 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.



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

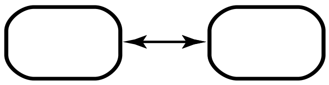
- You use a replica as a backup.
- Your company provides read-only use of your database to another site (or company).
- A high-security development project uses the same files as a more open project. In this case, the open project sends updates to the high-security project, but no updates are sent in the other direction.

Note: 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.

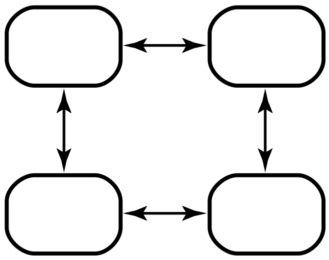
One-to-One and ring synchronization

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

One-to-one synchronization pattern



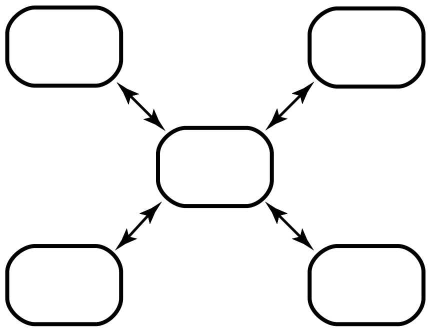
Ring synchronization pattern



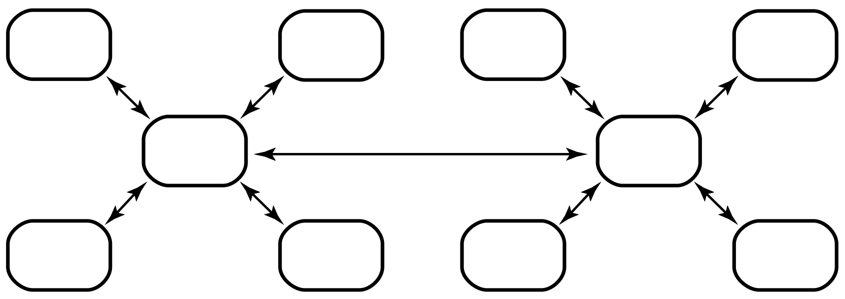
One-to-Many synchronization

There are several one-to-many commonly used synchronization patterns.

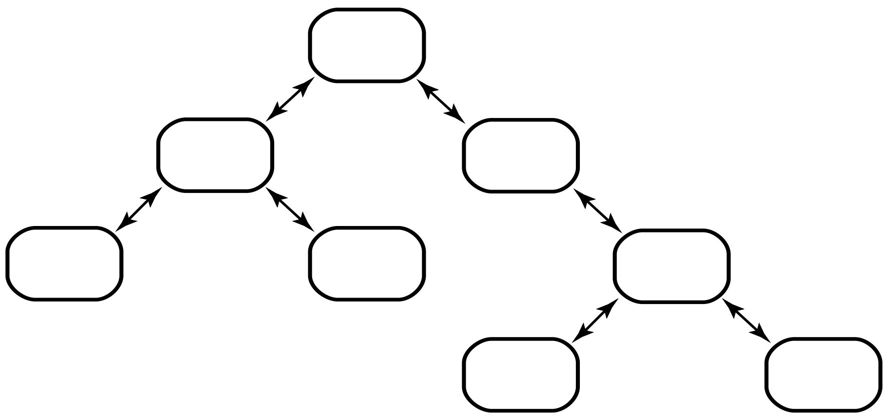
Single-hub synchronzation pattern



Multiple-hub synchronization pattern



Tree synchronization pattern



Advantages

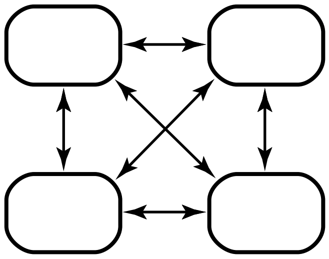
- More efficient for the spoke and branch replicas, which send to and receive from only one other replica.

Disadvantages

- If the hub or root site goes down, all spoke sites must reconfigure their pattern to keep communication going.
- If you change the synchronization pattern so that replicas that did not synchronize directly now exchange packets, the first packets that are generated may be too large for the system.

Many-to-Many synchronization

The many-to-many is a more complex synchronization pattern.



Advantages

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

Disadvantages

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

Synchronization schedule

The synchronization schedule for a replica 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 and the connections among sites.

Consider the following issues when planning your synchronization strategy:

- Rate of development.

  If you schedule synchronizations frequently, updates take less time as fewer changes have taken place. Data loss is kept to a minimum if a replica is deleted accidentally and you must restore it from backup.

- Time zone differences.

Defining mastership policies

With ClearQuest MultiSite, you need to take mastership policies into consideration when planning your defect-tracking processes. Mastership adds another layer of control within your process.

For example, as a defect moves from one state to another, different sites can be assigned mastership. Or you may choose to have all records of a certain type, regardless of state, mastered and modified at a specific site.

Mastership can effect different aspects of your defect-tracking process. For example:

- Hooks that modify records or field values cannot run if the current site does not have mastership of the record.
- Users and groups can only be modified at the site that currently has mastership of the user or group.
- Workspace items (queries, reports and report formats) can only be edited at the mastering replica of the query you want to modify.
- Only one site within a clan can modify or customize schemas.

Setting up mastership for records

ClearQuest MultiSite includes a system field, ratl_mastership, which tracks the mastership of a record. The value of this field is always the name of the replica which currently masters the respective record.

The mastership field can only be changed at the site of mastering replica. For more information about setting up mastership for records and detailed instructions on adding the field to the record form, see *Using mastership with records* on page 72.

Managing mastership changes

Mastership changes are communicated among replicas by the standard synchronization mechanism, see Chapter 4, *Synchronizing replicas*.

Depending on your workflow, mastership changes for some objects may need to occur more often. For example, the mastership of a record may need to be changed from one replica to another several times during its lifecycle.

To facilitate these mastership changes, you may want to consider ways to streamline the request for mastership process for records. You can:

- Write an e-mail rule that sends e-mail to the ClearQuest MultiSite administrator of the mastering replica when a change in mastership is needed.
- Allow other ClearQuest MultiSite administrators access to your replica via ClearQuest Web so they can logon and change the mastership field when needed.
- Call or e-mail the ClearQuest MultiSite administrator at the mastering site to ask that they change the mastership of an object.

Defining a transport mechanism

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

For detailed information about how to set up your transport mechanism, see Chapter B, *Configuring store-and-forward*.

Your situation	Recommended method	More information
Sites are connected with high-speed lines	store-and-forward	*Using Store-and-Forward* on page 102 **shipping_server** reference page
One or more sites have firewalls	E-mail or store-and-forward,	*Installing Store-and-Forward on a firewall host* on page 120 **syncreplica** reference page
Must transfer packets quickly	E-mail, store-and-forward or ftp	**shipping_server** reference page, **syncreplica** reference page MultiSite Control Panel reference page
No electronic connection between sites	external storage media such as diskette or CD	**syncreplica** reference page

Overview of deployment tasks

Before deploying ClearQuest MultiSite, you should test out your plan and do “test run” to ensure that your process, both sychronization and defect-tracking process is effective. For a detailed checklist, see *Sample deployment checklist* on page 35.

1 **Review ClearQuest MultiSite documentation, and install ClearQuest MultiSite.**

Remember that installing ClearQuest MultiSite involves a ClearQuest upgrade, for more information see the *Upgrading ClearQuest* chapter in the *Installing Rational ClearQuest* manual.

Note: All user databases associated with a schema repository must be upgraded to the same version of ClearQuest before you can begin using ClearQuest MultiSite.

2 **Plan out your replication strategy and mastership policies.**

Prepare a ClearQuest MultiSite Workflow document that describes the changes and policies that will be enforced. This plan should describe the workflow for users performing tasks in a replicated environment (when they'll need to request mastership of an object, what replica sites will manage which projects, when to expect synchronization updates, etc.) Representatives of the user community should review this.

3 **Decide how you will modify your schema and process to work with ClearQuest MultiSite.**

Prepare a document that describes the changes that are necessary to implement ClearQuest MultiSite (adding the mastership field to your schema, modifying existing hooks, creating hooks that automate transfer of mastership, etc.)

4 **Create a copy of your production database and then activate and replicate it.**

You can then implement your schema changes and test them within a replicated environment without disturbing your work environment. This process is similar to performing a test upgrade, for more information about making a copy of your database, see the Upgrading ClearQuest chapter of the *Rational ClearQuest Installation Guide*. Achieving this milestone ensures that the personnel are adequately trained and the resources are in place without disturbing your production database. Be sure to test your own backup and recovery processes on the replica.

5 **Test your new process and schema changes within the replicated copy of the database.**

In particular, you should test any new hooks and test that updates are being sent and received, using the parameters that you have set up. Remember that at this point, you will have two replicas, the MultiSite-activated copy of your production database and the replica you created of that copy.

6 **Review testing results and workflow changes.** This may result in additional activities such as additional changes to the workflow.

7 **Activate and replicate the production database.**

Set up unidirectional synchronization with your production database to test your synchronization scripts to ensure that no data is being lost.for a short period of time to ensure that synchronization is occurring correctly. At this point, users at the replica site are not using the replica. At this point the replica can be removed without loss of data if problems develop.

8 **Upgrade the production database with the new schema changes.** Schema changes are introduced into the production database (after having been tested in the copy).

9 **Users at the replica site can begin using the replica.** The new workflow rules are now in effect. It is also recommended at this point that a web server be set up for the new site to provide remote access for personnel at other sites.

Sample deployment checklist

Task	Reference	Planned	Actual
1 Staff trained and ClearQuest MultiSite installed	Administering ClearQuest MultiSite Installing ClearQuest		
2 Prepare ClearQuest MultiSite workflow document			
3 Prepare ClearQuest schema design document			
4 Create a copy of your production database	Installing Rational ClearQuest ClearQuest Import Tool help ClearQuest Export Tool		
5 Implement your schema changes in the copy of the production database	Administering ClearQuest, Chapter 5, Customizing a schema. Administering ClearQuest MultiSite, *Using mastership with records* on page 72		
6 Replicate the copy of the production database and send it to the test location	Administering ClearQuest MultiSite, *Creating Replicas* on page 37, *activate* on page 126, and *mkreplica* on page 162.		

Task	Reference	Planned	Actual
7 Test your synchronization process and schema changes in the replica, make any necessary changes	Administering ClearQuest MultiSite, *Using Store-and-Forward* on page 102		
8 Activate the production database	Administering ClearQuest MultiSite, *Creating Replicas* on page 37 and *activate* on page 126		
9 Replicate the production database and send it to the intended site(s)	Administering ClearQuest MultiSite, *Creating Replicas* on page 37 and *mkreplica* on page 162.		
10 Implement your schema changes in the production database and send an update packet to update the new replica	Administering ClearQuest, *Chapter 5, Customizing a schema*. Administering ClearQuest MultiSite, *Using mastership with records* on page 72 and *syncreplica* on page 207		
11 Start using the replica			

Creating Replicas 3

This chapter describes how to create database replicas. The topics covered include:

- Understanding the replication process
- Activating the database
- Exporting a replica
- Importing a replica
- Replica-Creation scenario
- Creating a replica using store-and-forward

Understanding the replication process

Replicating a database involves tasks on both the site that is replicating the database (exporting) and the site that will be receiving (importing) the replica.

At the replicating (exporting) site, the administrator must do the following:

1. If using store-and-forward, configure the MultiSite Control Panel to store, send, and receive replica packets, see Appendix B, *Configuring store-and-forward*.
2. Activate the respective database set to MultiSite status using **activate**. This step is only required when making the initial replica of a clan.
3. Replicate the database using **mkreplica -export**.

At the receiving (importing) site, the administrator must do the following:

1. If using store-and-forward, configure the MultiSite Control Panel to store, send, and receive replica packets, see Appendix B, *Configuring store-and-forward*.
2. Create empty vendor databases for the schema repository and each respective user database replica.

3 Import the replica using **mkreplica -import**.

Commands	When used:
activate	Before you can create the first replica of a clan. This command only needs to be done once,
mkreplica	To create and import replica packets. Administrators must first create a physical database to which to import a replica packet.

Activating the database

Before you can create the initial database replica of a clan, you must first activate the database set (a schema repository and its associated user databases). Once a database set has been activated, multiple families and sites can be replicated. A database set only needs to be activated once.

When you activate a database set to be replicated, you assign it a clan and site name and indicate the host where its update packets will be sent and received.

If you want to allow users to change the mastership of database records from the ClearQuest client, you need to make the appropriate schema changes and upgrade the respective user databases *before* you activate the database set (see Chapter 6, *Managing mastership*).

You activate a database set to be replicated with the **activate** command. For more information about activating a database, see *activate* on page 126.

Exporting a replica

Replicas are made by creating replica packets and then sending them via the network or other transport method to the site which will host the database replica. This is done with the mkreplica -export command. For more details about using **mkreplica -export**, See the mkreplica reference page on page 162.

After a database set has been activated, you use this three-phase procedure to create new database replicas:

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

The basic procedure is the same for all methods of packet delivery and for all platforms.

Note: You cannot replicate test databases.

During the export phase of replica creation, the replica creation command locks the database while copying it. The database is locked for the entire length of time the command runs and logins are not allowed.

Before running the **mkreplica -export** command on a database that has not been replicated, you MUST make sure that no users are currently logged into the database. If users are logged into the database during a **mkreplica -export**, any changes they make even after the database is unlocked will be LOST.

The time needed to create the packet depends on the size of the database and can take up to twice as long as it would to make a copy of the database or run a backup procedure.

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

With this example, a replica of the PRODA user database and its respective schema repository was made for the paris site which uses the host machine, server3. This example uses Store-and-Forward to ship the replica-creation packets.

```
multiutil mkreplica -export -clan telecomm -site boston
-family PRODA -user admin -password secret -maxsize 50m
-fship -workdir c:\temp\packets -sclass cq_default
server3:paris
```

Importing a replica

When you import a replica, you import raw data into existing vendor database(s). Replica packets do not contain databases, they contain the metadata and record data that form a database. Packets are not vendor-specific and can be used to create a ClearQuest database from any supported vendor database.

When you import a replica, remember the following:

- You must use the same site name as was given to the replica when it was exported. You cannot change the site name when you import a replica.
- Initially, you can only access the replica with multiutil commands from the same machine where you first ran **mkreplica -import**. If you want to use a different machine to run subsequent multiutil commands, you'll need to configure that machine to access the replica with multiutil, see *Running multiutil commands at multiple machines at the same site* on page 19.

To receive (import) a replicated database at your site, you must do the following:

1. If using the Rational Shipping Server, configure the MultiSite Control Panel or the shipping.conf (UNIX) to store, send, and receive replica packets, see Chapter B, *Configuring store-and-forward.*
2. Create at least two empty vendor database to which to import the replica data, see *Installing Rational ClearQuest*.
3. Import the replica by running the **mkreplica -import** command.

Creating empty vendor databases

Before receiving and importing a database replica, you need to create an empty vendor database(s) in which to import the replica data. See *Installing Rational ClearQuest* for further instructions on creating vendor databases for ClearQuest to use.

Note: ClearQuest MultiSite does not support Microsoft Access or SQLAnywhere.

Warning: Do NOT create a ClearQuest database before receiving a database replica. A ClearQuest database is automatically created when you import a replica packet into an empty vendor database. If you have created a ClearQuest database with the Maintenance Tool or ClearQuest Designer, the replica import will fail.

Importing the replica

After verifying that you've received a replica packet, you need to run the **mkreplica -import** command to import the replica data into the empty vendor database you've created. You'll need to enter the database parameters and login information for both the schema repository and user database you are importing.

Adding additional replicas at the same site

If you have an existing replica at your site and the replica you want to import is of the same clan, but a a different family (it originates from the same working schema repository), you do not need to create a vendor database for the schema repository. If the incoming replica is of the same clan, the **mkreplica -import** command automatically associates the new replica with the existing replicated schema repository. See the mkreplica reference page on page 162.

Recovering from a failed replica import

If the replica import process is interrupted or fails for any reason, do the following:

1. Verify when the import failed. mkreplica -import generates error messages containing this information.
2. Delete the vendor database where the import failed.
   - If the import failed during the import of the replicated schema repository, you'll need to delete the vendor database used for the replicated schema repository.
   - If the import failed after a successful import of the replicated schema repository, you'll need to delete the vendor database used the user database replica.
3. Create a new vendor database.
   - If the import failed during the import of the replicated schema repository, you'll need create a new vendor database in which to import the replicated schema repository.
   - If the import failed after a successful import of the replicated schema repository, you'll need create a new vendor database in which to import the user database replica.
4. Run **mkreplica -import** again.

Replica-Creation scenario

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

Relevant characteristics of the two replicas:

Replica Site location	Host name	Replica name
Boston	minuteman	boston_hub
San Francisco	goldengate	sanfran_hub

Prerequisites

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

1. Make sure ClearQuest MultiSite licenses are installed.

   After you enter the **activate** command on a user database, users at the original site cannot access the database without a ClearQuest MultiSite license (in addition to a ClearQuest license).

2. When creating a replica of a specific database for the first time, all users must log off of the database.

   The **mkreplica -export** command locks the database after you start to export the replica. All users should logoff before the procedure begins and log back on after it is finished. Data is lost if ClearQuest sessions are left open during the replication process.

3. Determine the size of the user database and schema repository.

   Replica-creation packets can be up to five times larger than the respective databases. Be sure the working directory you use has enough free space. You must have write permission for the directory, and the directory you specify must not exist.

Activating the database set

1 In Boston, activate the database set.

The following example activates the database set (schema repository and its associated databases) at the Boston site. It names the clan (telecomm) and the site (sanfran_hub) and associates the host machine, minuteman. The host machine is where the storage bays for outgoing and incoming update packets are stored.

```
multiutil activate -user bostonadmin -password secret -clan
telecomm -site boston -host minuteman
```

Export phase

At the Boston (originating) site, perform the following steps.

1 Use the **mkreplica -export** option to create the replica for the Boston site. See the mkreplica reference page on page 162 for information about restrictions on the command.

The following example creates the sanfran_hub replica of the PRODA user database that participates in the telecommunications clan. The storage bays for this replica reside on the goldengate host machine.

Also, the administrator uses the **–fship** option which sends the packet immediately, using the Rational Shipping Server.

```
multiutil mkreplica -export -clan telecomm -site boston
-family PRODA -user bostonadmin -password secret -maxsize 50m
-fship -workdir c:\temp\packets -sclass cq_default
goldengate:sanfran_hub
```

2 Back up the original database.

This backup records the fact that the database is replicated. If you have to restore a database replica from a backup copy that was made before the database was replicated, the ClearQuest 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 database is marked as unreplicated.)

Transport phase

1 Send the replica-creation packet to the new site. This process differs depending on the options you used during the export process:

- If you used **–fship** in Step 1, 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:

  ```
  shipping_server -sclass cq_default
  shipping_server -poll
  ```
- If you used **–out** to write the packet to a file, you must transport the file to the new site using the media of your choice.

Import phase

These steps take place at the San Francisco (receiving) site. The Boston site has no existing replicas. The San Francisco site administrator needs to have created empty databases to which to import the incoming replicas of the schema repository and user database.

1 If using store-and-forward, verify the packet's arrival by entering the **lspacket** command on the receiving host.

```
multiutil lspacket -short
Multiutil:Packet'd:\temp\ms_ship\incoming\mk_BOSTON_21-May
-01_19-28-01.xml'...
```

2 Create empty vendor databases. The San Francisco site administrator needs to have created empty databases to which to import the incoming replicas of the schema repository and user database.

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

In the **mkreplica –import** command, you must specify the pathname of the incoming packet as listed by the **lspacket** command. For example:

```
multiutil mkreplica -import -site boston -repository ORC1
-vendor ORACLE -dbologin orcadmin password -connectopts
host=boston_dbserver;SID=ORC1;server_ver:8.1;client_ver=8.
0;log_type=long -database ORC1 -vendor ORACLE ORC1
-dbologin orcuser password -connectopts
host=boston_dbserver;SID=ORC1;server_ver:8.1;client_ver=8.
0;log_type=long -comments ‘’Importing the initial replicas
of the PRODA database and its schema repository for the
Boston site in the telecommunication clan’’
d:\temp\ms_ship\incoming\mk_BOSTON_21-May-01_19-28-01.xml
```

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

5 Begin development.

Users in Paris can access the new replica in the same way they would access an unreplicated database.

Creating a replica using store-and-forward

If your sites have a high-speed connection, you can take advantage of the ClearQuest MultiSite store-and-forward facility when you create a new replica. If your current site does not have IP connectivity to the site of the new replica, you can use magnetic tape or a file-based packet transfer method like ftp or e-mail.

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

Communication between replica hosts

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

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

Limiting the size of a packet

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

```
multiutil mkreplica -export -clan telecommunications -site
boston -family PRODA -user bostonadmin -password secret
-maxsize 50m -fship -workdir c:\temp\packets -sclass
cq_default goldengate:sanfran_hub
```

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

Synchronizing replicas 4

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

- Understanding sychronization
- Planning synchronization
- Applying packets that include schema updates
- Synchronizing a replica using store-and-forward
- Synchronization scenario

Understanding sychronization

The sychronization process involves the same export-transport-import procedure that is used during replica creation:

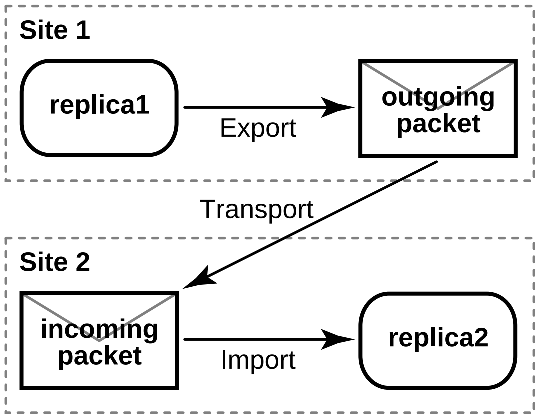
- **Export phase**—At one site, a **syncreplica** (synchronize replica) command is invoked with the **–export** option. This creates a *packet* of data.
- **Transport phase**—The packet is sent to one or more other sites.
- **Import phase**—At the other sites, a **syncreplica** command is invoked with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

Note that each synchronization is one-way. If two replicas update each other, two synchronizations are required.

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, based on its estimates of what the target replica has already been sent.

Command	When used
syncreplica	To create update packets to send out and to import update packets from other replicas.
chepoch	Used when you need to recover from a lost packet. Allows you to re-create a packet that has already been sent. Epoch numbers are automatically updated regardless of whether or not a packet is sent and applied successfully. If a packet fails, you'll need to reset the your replica's epoch estimate of the replica where the sychronization failed.
lsepoch	Used when you need to recover from a lost packet. Allows you to list the current epoch estimates for the replicas within a family. You can then determine which epoch number to use if you need to reset an epoch estimate.

The following illustrates the ClearQuest MultiSite replica-synchronization scheme. At Site 1, a **syncreplica –export** command places database update information from **replica1** into a packet. The packet is sent to Site 2. At Site 2, a **syncreplica –import** command imports the contents of the packet into **replica2**.



Planning synchronization

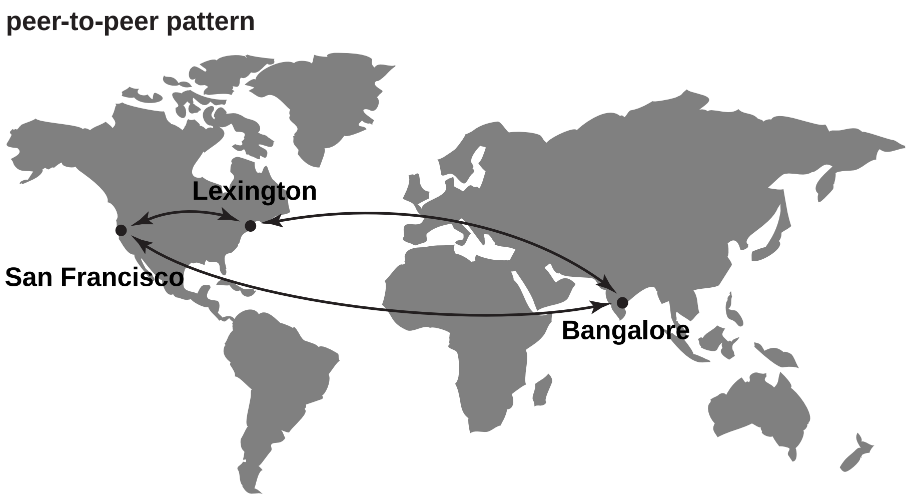
The following example depicted a simple case, involving one point-to-point update. All updates need not be point to point, however, because they are cumulative. Suppose that the following updates take place among three replicas:

- Update 1—Replica 1 sends changes to Replica 2
- Update 2—Replica 2 sends changes to Replica 3

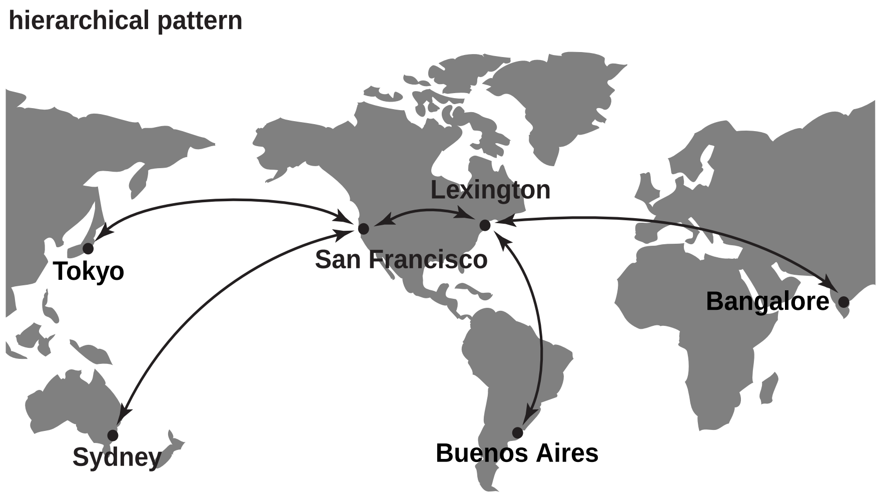
There is no need for Replica 1 to update Replica 3 directly, because the changes from Update 1 are included in Update 2. This feature gives administrators flexibility in devising update strategies and patterns. For efficiency, a single update can be targeted at multiple sites, for example, all other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

In general, you can implement any update topology, as dictated by organizational structures, communications/transportation costs, and so on.

peer-to-peer synchronization update pattern



double-hub hierarchical update pattern



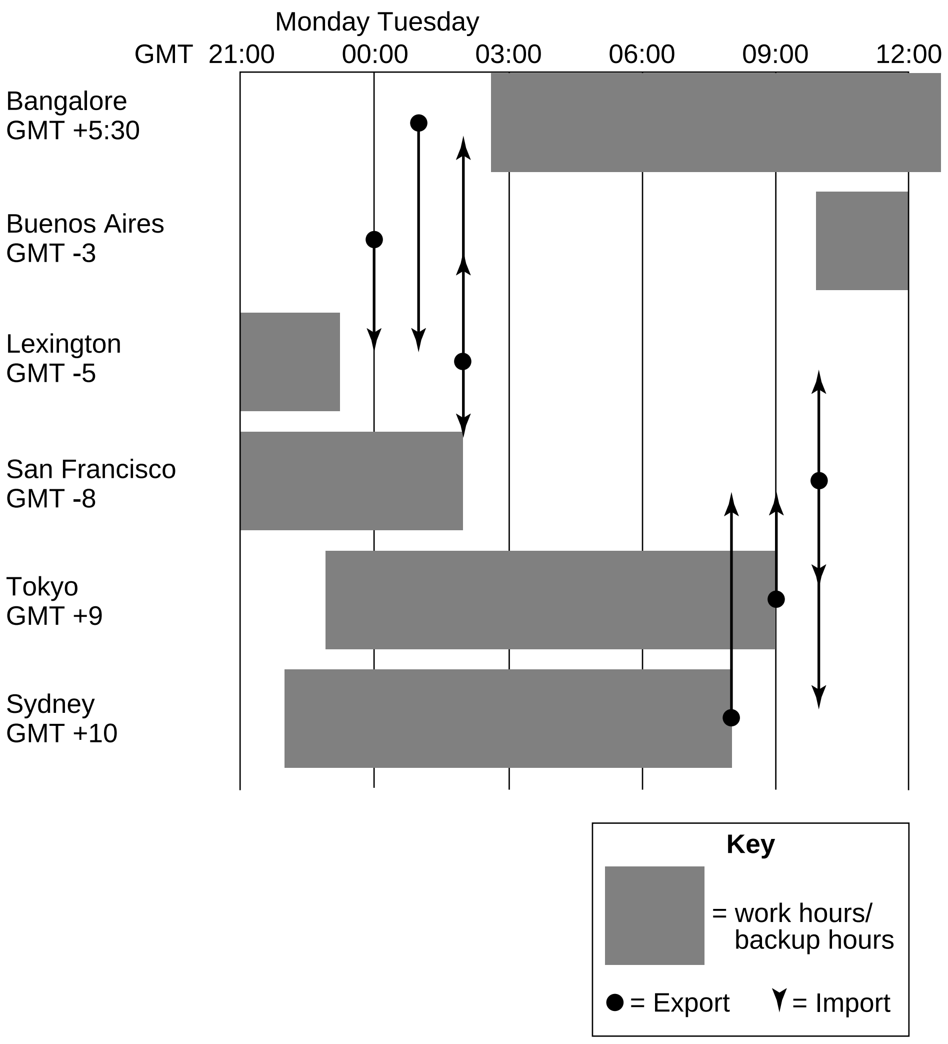
Designing an update strategy

Site administrators must design a strategy for sending updates among the various replicas. They must specify an update pattern for the database family and an update frequency for each replica.

For example, the administrators for the database family in shown in the hierarchical pattern figure make the following decisions:

- The hub replicas, which undergo rapid development, synchronize every hour.
- Each hub replica synchronizes daily with its spoke replicas. Each spoke replica will send an update packet to the hub replica, and then the hub replica will send update packets back to the spoke replicas. Because these packets may be large and take a long time to import, the synchronization should not take place during working hours.
- All sites use receipt handlers to import packets as soon as they are received.

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



Assumption of successful synchronization

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

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

Applying packets that include schema updates

Packets originating from the working schema repository may include new schema revisions that require database upgrades at respective family sites. In this case, the **syncreplica -import** cannot finish its process until the user database replica is upgraded to the new schema version.

If a packet contains updates to both the replicated schema repository and the user database replica, syncreplica -import stops the process and gives the following message:

```
<packet_name> is destined for schema revision
<revision_number>, not <revision_number>; re-execute
syncreplica after site admin has upgraded database.
```

In this case, you'll need to upgrade the affected user database replica and then run **syncreplica -import** again.

If you've automated your synchronization process, the automation script will fail and additional packets dependent on the schema changes cannot be applied.

Synchronizing a replica using store-and-forward

If your sites have a high-speed connection, you can take advantage of the ClearQuest MultiSite store-and-forward facility when you synchronizing replicas. If your current site does not have IP connectivity, you can use magnetic tape or a file-based packet transfer method like ftp or email.

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

Communication between replica hosts

The hosts must be able to communicate with each other. If your network uses host names, the sending host must be able to resolve the receiving host's name to an IP address. To accomplish this, you may have to update the Domain Name Service.

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

```
multiutil mkreplica -export -clan telecomm -site boston
-family PRODA -user bostonadmin -password secret -maxsize 50m
-fship -workdir c:\temp\packets -scl cq_default sanfran_hub
```

For information on default packet size limits, See the syncreplica reference page on page 207.

Transport options

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

The outgoing packet is stored in the **outgoing** subdirectory of a storage bay.

The **incoming** and **outgoing** subdirectories of storage bays contain packets waiting for transport or processing. All shipping operations look for packets in these subdirectories.

- At the receiving site, the incoming packet is stored in the **incoming** subdirectory of a storage bay.

Automating synchronization

You can automate all phases of synchronization with the use of third-party tools:

- **Export phase**. Create a script that runs at scheduled times that creates and sends update packets from one or more replicas at the site to one or more siblings.
- **Transport phase**. The *store-and-forward* facility handles packets of any size. You can invoke store-and-forward as part of the export phase, or automate packet transport separately.
- **Import phase**. A ClearQuest MultiSite receipt handler runs whenever a packet is received at a replica.
- For more information about using receipt handlers and to see a sample script, see *Automating synchronization* on page 110.

Synchronization scenario

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

Export phase

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**):

```
multiutil syncreplica -export -clan telecomm -site
sanfran_hub -family PRODA -user sfadmin -password secret
-maxsize 50m -workdir c:\temp\packets -fship -sclass
cq_default bangalore paris
```

Transport phase

Send the packets. If you did not use, **–fship**, Use electronic mail, regular mail, or your preferred delivery method. If you used **syncreplica –export –ship**, invoke **shipping_server** in either of the following ways:

```
shipping_server -poll
shipping_server \\goldengate\shipping\outgoing\
```

Import phase

1. **(If you used diskettes or electronic mail) Copy the packet files into a directory.**
2. **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.

   ```
   multiutil syncreplica -import -family PRODA -user parisadmin
   -password secret -receive -scl cq_default
   ```

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

   ```
   multiutil syncreplica -import -family PRODA -user parisadmin
   -password secret \\baguette\shipping\incoming\
   ```

Managing replicas 5

This chapter describes how to manage existing replicas, including how to delete a replica. For information on creating a replica, see Chapter 3, *Creating Replicas*. For information on enabling requests for mastership in a replica, see Chapter 6, *Managing mastership*.

The following topics are covered in this chapter:

- Understanding replica management
- Displaying properties of a replica
- Changing the host name for a replica
- Moving a replica
- Removing a replica
- Scrubbing parameters for database replicas

Understanding replica management

Replica management involves replica administration tasks that are in addition to any database administration tasks such as creating backups. When working with a database replica, tasks such as moving a database become more complicated as MultiSite settings may need to change.

Commands	When used:
chreplica	To change the location of the host that delivers update packets to a replica. Use the chreplica command to change the host properties.
lsreplica	List all replicas in a family or a clan.
restorereplica	Automatically brings a replica up-to-date after it has been restored from a vendor database backup. You'll need to use this command to request packets from one or more sibling replicas that will include the data necessary to bring the restored replica up-to-date.
rmreplica	When you want to decommission a replica and inform other replicas not to expect additional updates. Before you can remove a replica, you'll need to transfer the mastership of all objects that it masters to another replica(s).

Displaying properties of a replica

The **lsreplica** command displays the properties of a replica. For more information see *lsepoch* on page 148.

For example, to display the names of all replicas in the PRODA family:

```
lsreplica -clan telecomm -site paris -family PRODA
-user parisadmin -password secret -short
BANGALORE
BOSTON
PARIS
SANFRAN
```

Changing the host name for a replica

When you move the Rational Shipping Server and the MultiSite Control Panel or when you rename a replica's host, you must update the host name in the Control Panel. The replica keeps track of the hosts on which the replicas in a database 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. The change is not propagated to other replicas in the database 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:

```
multiutil chreplica -clan telecomm -site paris -family PRODA
-user parisadmin -password secret -host pserver2 paris
```

Moving a replica

You can move each of the three components of a replica site:

- the host
- the user database replica
- the schema repository replica

There are some special considerations when you move a replicated database or its associated shipping server host:

- Make sure ClearQuest MultiSite is installed on the new shipping server host.
- If you automated the synchronization process on the old host, you must set up synchronization export and import scripts on the new host.
- After physically moving a user database replica, use ClearQuest Designer to tell ClearQuest where to locate the database (**Database > Update User Database Properties or Database > Move User Database**).
- After moving the replica, export update packets to all sibling replicas.

After moving the database replica, change the host name associated with the replica by using **multiutil chreplica –host**. You must enter this command at the mastering replica of the replica you moved.

Moving a shipping server host

There may be times where you want to use a different machine to send and receive update packets. For example, the host machine you were using has had a hardware failure and needs to be replaced.

In this case, use the following steps:

1. Install ClearQuest MultiSite and the Rational Shipping Server on the new machine.
2. Use the chreplica command to associate the new host name with the replica.

```
chreplica -clan telecomm -site bangalore -family PRODA -user
bangadmin -password secret -host server3 bangalore
```

Moving a user database replica

There may be times where you want to move your user database replica to a different location on the network or switch it to use a different vendor database software.

To move a user database replica:

1. Follow the instructions in the *Administrating Rational ClearQuest* manual (located in the \\ClearQuest\books directory) when you move your user database. A user database replica can be moved just as a non-replicated database.
2. Any host info that changed should be updated using **chreplica**.

Moving a replicated schema repository

There may be times where you want to move your replicated schema repository to a different location on the network or switch it to use a different vendor database software.

To move a replicated schema repository:

1. At a command prompt, set the dbset environment variable. You do this by typing:
   ```
   c:\Program Files\Rational Software\ClearQuest\set
   BB_TEST_DBSET_NAME=<database_set_name>
   ```
   where *`<database_set_name>`* is the name of the database set that includes the replicated schema repository. The database set name is determined when the initial schema repository was activated and uses the following format: `CQMS.CLAN_NAME.SITE_NAME .`
2. From the same command window as was used in Step 1, type `cqdbsetup` and press ENTER to open the ClearQuest Maintenance Tool.
3. Move the schema repository using the ClearQuest Maintenance Tool, using the instructions in the *Administrating Rational ClearQuest* manual (located in the \\ClearQuest\books directory. A replicated schema repository can be moved just as a non-replicated schema repository.

Removing 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 database family.

When you remove a replica using the rmreplica command, you effectively tell all replicas within its family not to expect any more update packets or to track epoch estimates for that replica. Removing a replica does not delete the database.

Removing a replica requires two synchronization cycles, one to transfer mastership of all of that 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 only be communicated using the sychronization process, a replica cannot remove itself because doing so prevents it from creating update packets.

Once a replica is removed from a family, it no longer participates in sychronization activities and ClearQuest MultiSite information is not tracked. It no longer keeps an oplog and it cannot transfer mastership of any object, including records.

To remove a replica, you must do the following:

1. Transfer mastership of all objects to another replica.
2. Export and send an update packet from the replica to be removed.
3. Have any other replica run **rmreplica** command at their site to remove the respective replica
4. Export and send an update packet from the replica which removed the respective replica.
5. Run **rmreplica** at the site which is to be removed.

The replica is effectively removed after all replicas have been synchronized.

Note: If a replica is deleted mistakenly and you want to restore it from backup, see *Restoring replicas* on page 97. If a replica's storage directory is lost and there is no backup, see *Cleaning up from accidental deletion of a replica* on page 100.

Remove replica scenario

In this scenario, the replica **tokyo** in the database family **PRODA** is being removed.

1 Transfer mastership of all objects to another replica.

a At the site of the replica to be removed, transfer mastership of all objects mastered by the replica to another replica.

In this example, the administrator transfers mastership of all of it's ClearQuest objects, including records to the Boston replica:

```
multiutil chmaster -clan telecomm -site toyko -family
PRODA -user bostonadmin -password secret boston -all
```

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

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 a user or group* on page 77. If there are problems you cannot fix, another replica can recover from the error by assuming mastership of the objects. For a description of this procedure, see *Cleaning up from accidental deletion of a replica* on page 100.

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

The replica to be removed must send its final changes, including any new records and mastership changes, to the replica receiving mastership. The replica to be removed can broadcast its final changes to all other replicas, but it must update the new mastering replica (**boston** in this example).

```
syncreplica -export -clan telecomm -site toyko -family PRODA
-user toykoadmin -password secret -workdir
e:\temp\syncstorage -fship -sclass cq_default boston
```

3 Import the update packet at the replica that is (or will become) the mastering replica of all the ClearQuest objects that are mastered by the replica to be removed.

```
syncreplica -import -clan telecomm -site boston -family PRODA
-user bostonadmin -password secret -receive -sclass
cq_default
```

4 At another replica, remove the replica. It is recommended that you do this at the working schema repository site.

```
multiutil rmreplica -clan telecomm -site boston -family PRODA
-user bostonadmin -password secret toyko
```

5 At the replica where you removed the respective replica, export and send an update packet to the remaining replicas in the family.

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

```
syncreplica -export -clan telecomm -site boston -family PRODA
-user bostonadmin -password secret -workdir
e:\temp\syncstorage -fship -sclass cq_default sanfransisco
bangalore
```

6 Optionally, at the replica that was removed, run **rmreplica**.

```
multiutil rmreplica -clan telecomm -site toyko -family PRODA
-user toykoadmin -password secret toyko
```

Removing the last replica of a clan

If you decommission and remove all replicas, the one remaining replica (the working schema repository) can be switched to a regular ClearQuest database (a non-replicated database and schema repository), and developers no longer need a ClearQuest MultiSite license to access it.

In this case, you'll need to run the **-rmreplica** command at the site you want to remove and use the **-dbset** argument to change the database set name associated with the schema repository and it's user database(s). There are no mastership concerns because the database will no longer be replicated.

For example:

```
multiutil rmreplica -clan telecomm -site boston -family PRODA
-user bostonadmin -password secret -dbset maindatabase
```

Scrubbing parameters for database replicas

When a ClearQuest action or ClearQuest MultiSite command makes a change to a replica, an *oplog entry* is recorded in the replica's database. (See *Database operations and the oplog on page 8* for more information on this mechanism.) Also, when you export an update packet, an export_sync record is created for each target replica. These records are stored in the database replica and are used by the **recoverpacket** command to reset a replica's epoch number matrix.

You can scrub oplog entries and export_sync records to reclaim disk space, 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.

Oplog scrubbing

Oplog entries must be kept in the database for a significant period. In the near term, they are required when the replica generates update packets to be sent to all other replicas. Beyond that, entries may be required to help other replicas recover from catastrophic failures. If no replica can supply these entries, the replica being restored must be re-created. 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. When it is safe to delete oplog entries for a replica, follow these steps:

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

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

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

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

2 Use the **scruboplog** command at the replica whose oplog you’d like to reduce.

This example scrubs the oplog of the user database replica (indicated by the PRODA family) at the *sanfrancisco* site:

```
scruboplog -clan telecommunications -site sanfrancisco
-family PRODA -user sfadmin -password secret -before 10/31/01
```

Managing mastership 6

This chapter describes how to manage the mastership of ClearQuest objects in a database replica, and covers the following topics:

- Understanding mastership
- Using mastership with records
- Using the chmaster command to change object mastership
- Forcing a transfer of mastership
- Fixing an accidental mastership change

Understanding mastership

In a MultiSite environment, tracking changes and preventing data corruption is accomplished with an exclusive-right-to-modify, called *mastership*. Mastership determines when a user of a database replica is allowed to modify data. For example, without using mastership, users could modify records in different replicas independently and simultaneously, the result would be chaos. Suppose a record SAMPL00001 was modified in three replicas at the same time. Which is the real record SAMPL00001, and what ought to happen to the other two versions?

With mastership, database records and other ClearQuest objects (records, schemas, queries, etc.) are assigned a *mastering replica*. The initial mastering replica of a ClearQuest object is the site where the object is created. The mastering replica can be changed subsequently. When you create an object in a replicated database, your current replica is mastering replica of the new object.

The following ClearQuest MultiSite objects have mastership properties that can be transferred between replicas:

- Records
- Workspace items
- Users and groups
- Schema repositories

Command	When used:
chmaster	Used to change the mastering replica of a particular object. You can only change the mastership of an object from the mastering replica of that object.
describe	To find out which replica currently has mastership of the respective ClearQuest object.

Changing mastership

You can transfer mastership of an object by an explicit chmaster command or in the case of records, you can use the ratl_mastership field, see *Using mastership with records* on page 72. Some examples of when mastership changes are appropriate:

- You want to make changes to user information that is mastered at a different site.
- You want to allow another site to modify a public query
- Before you decommission a replica, you must transfer mastership of each object mastered by that replica to one of the remaining replicas. (See *Removing a replica* on page 64.)

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

1. Notify the administrator of the mastering replica that you need mastership of a database object.
2. The administrator at the mastering replica changes mastership of one or more objects to another replica.
3. The administrator at the mastering replica site exports and sends an update packet from the old mastering replica to the new mastering replica.
4. Import the update packet at the new mastering replica.

   Until the update packet containing the mastership change is imported at the new mastering replica, mastership is “in the packet” and the replicas in the database family have different information about which replica masters the object.

   For example, the administrator at the **sanfran_hub** replica transfers mastership of the user group “QA_ENGINEERING” to the **bangalore** replica, and then exports an update packet. At this point:

   - The **sanfran_hub** replica considers the user group to be mastered by **bangalore**.
   - The **bangalore** replica considers the user group to be mastered by **sanfran_hub**.

- No one can modify the user group.

When you complete the mastership transfer by importing the update packet at **bangalore**, users at **bangalore** are able to modify the user group “QA_ENGINEERING”.

Using mastership with records

ClearQuest MultiSite includes with a system field called **ratl_mastership** whose value reflects the current mastering replica of a record. In other words, the value of the **ratl_mastership** field refers to the name of the replica which currently masters the respective record.

To allow users to change the mastership of database records, you’ll need to add this field to the respective record form.

The **ratl_mastership** field provides great flexibility in controlling when users can change the mastership of a record. Remember that the mastership field can only be changed at the site of mastering replica. Record mastership scenarios include, but are not limited to:

- Unrestricted ability to change mastership. All users can modify the **ratl_mastership** field and thus change the mastership of any record.
- Restricted ability to change mastership. You can make the mastership field read-only to specific users and groups.
- Hook-driven mastership. You can write a field hook that automatically changes the value of the **ratl_mastership** field (thus the mastering replica) when certain criteria are met.

Understanding MultiSite system fields and record types

The **ratl_mastership** field is a reference field that points to the replica record type. The replica record type is a stateless record type that automatically tracks all replicas in a family. Whenever a new replica is added to a family, it’s information is stored as a replica record. The replica record type and all replica records are read-only. You can use ClearQuest to query on the replica record type to find out more about the replicas that participate in a given family.

For a complete list of the system fields used for ClearQuest MultiSite, see *Administering Rational ClearQuest, Appendix A*.

Allowing users to access the ratl_mastership field

The **ratl_mastership** field automatically contains the value of the current mastering replica of a record. To allow users to change this value (thus giving mastership of a record to another replica), you must add the field to an existing form on the respective record type.

By allowing users to access this field (adding it to your record form), any ClearQuest users can change the mastership of any record, if they are at the mastering replica site of respective record.

To do this, you must add the field to the record form of each respective record type. Schema changes must be done at the working schema repository and are sent as update packets to members of the respective replica sites.

You can add the **ratl_mastership** field to the schema used by a family at any time.

For each record type in your schema for which you want to track mastership, use the ClearQuest Designer modify the schema of the respective replica family. You must do this at the site of the working schema repository.

1. Select a record type to which to add the field.
2. Choose a form to which to add the field.
3. Double-click the form to display it.
4. Select the tab to which you want to add the field. You can either add the field to an existing tab on the form or add a new tab to contain the field.
   - Select **Edit > Add Tab** to add a new tab.
5. Using the Field List, drag-and-drop the ratl_mastership field to the appropriate tab.
6. Check-in the modified schema.
7. Upgrade the appropriate user database to use the new schema.
8. Generate an update packet. Update packets automatically contain schema updates, if any. Keep in mind that each site will need to upgrade their replica to use the new schema revision see *Applying packets that include schema updates* on page 55.

Alternative ways to change record mastership

Depending on your process, the mastership of a record may change times many times during its life cycle. Although mastership changes must always be propagated through a synchronization cycle, there are ways to shorten the turnaround time in which mastership changes can take place.

- Increase the timing of your synchronization cycles.

  Synchronization cycles can take place as often as every five minutes. The more often you synchronize, the quicker mastership changes can take place.

- Configure a ClearQuest Web server for your replica.

  This allows other sites to remotely access your replica(s) instead of contacting your to request mastership. Users can use the ClearQuest Web to change mastership at the remote replica, while keeping the performance advantages of using ClearQuest MultiSite for editing local records.

- Combining quick synchronization cycles with the use of ClearQuest Web to access remote replicas.

Using the ratl_mastership field to change record mastership

With the ratl_mastership field, record mastership is changed with the following steps:

1. A user at Site A modifies SAMPL0001 and changes the value of the ratl_mastership field from Site A to Site B.
2. During the next sychronization, Site B's replica is updated and is given mastership of SAMPL0001. Remember that until the next sychronization, mastership of the record is "in the packet" and neither site can modify the record.

Using the chmaster command to change object mastership

The following ClearQuest MultiSite objects have mastership properties that can be transferred between replicas:

- Records
- Workspace items
- Users and groups
- Schema repositories

Mastership properties are transferred using the chmaster command, for more information and additional examples, see *chmaster* on page 131.

Transferring mastership of a record

When you use the chmaster command, only ClearQuest users with the Super User privileges and access to MultiSite administration tools can change the mastership of a record. In addition, mastership can only be changed at site of the mastering replica.

Because of the dynamic nature of database records, ClearQuest MultiSite provides two ways to change the mastership of a record. You can change the mastership of database records by using the chmaster command or by using the ratl_mastership field. For more information about record mastership, see *Using mastership with records* on page 72.

When you create a new record, it is mastered by the replica in which you create it.

To transfer mastership of an record to another replica using the chmaster command, follow these steps:

1. At the mastering replica (boston), enter a **chmaster** command:
   ```
   multiutil chmaster -clan telecomm -site boston -family PRODA
   -user bostonadmin -password secret bangalore
   entity:PRODA00013
   multiutil: The mastership of entity:PRODA00013 has been
   changed to site 'banaglore'
   ```
2. At the old mastering replica, export an update packet to the new mastering replica's site:
   ```
   multiutil syncreplica -export -clan telecomm -site boston
   -family PRODA -user bostonadmin -password secret -workdir
   d:\shipping\temp -fship -sclass cq_default bangalore
   ```
3. At the new mastering replica, import the packet:

```
multiutil syncreplica -import -clan telecomm -site bangalore
-family PRODA -user bangadmin -password secret -receive
-sclass cq_default
```

4 At the new mastering replica, verify that mastership has been received:

```
multiutil describe -clan telecomm -site bangalore -family
PRODA -user bangadmin -password secret entity:PRODA00013
multiutil: The mastership of entity:PRODA00013 is ‘banaglore’
```

Transferring mastership of a Workspace item

This section describes how to change mastership of a Workspace item using the **chmaster** command.

When working with Workspace items, the entity-selector format is `workspace:<query_name>`.

Use the following guidelines when denoting Workspace items:

- Quotes are used because the folder name includes spaces.
- In many cases, Workspace item names are case-sensitive, so follow the same case as was used in the ClearQuest Workspace.
- Include the full path name (folders and subfolders) of the Workspace item.

  For example, to list the query that is found in the Personal Folder\Projects folder, you’ll need to type the following:

  ```
  workspace:”Personal Folders\My Projects\My Query”
  ```

  To change the mastership of a query found in the Personal Folder of a user other than the one currently logged in and using multituil, you’ll need to type the following:

  ```
  workspace:”Personal Folders(user_name)\My Projects\My Query”
  ```

Only ClearQuest users with the Super User privileges can change the mastership of an object. In addition, mastership can only be changed at the site of the mastering replica.

Example of changing mastership of a Workspace item

1 At the mastering replica (boston), enter a **chmaster** command:

```
multiutil chmaster -clan telecomm -site boston -family PRODA
-user bostonadmin -password secret bangalore
workspace:”Public Folders\Triage\project report”
```

2 At the old mastering replica, export an update packet to the new mastering replica’s site:

```
multiutil syncreplica -export -clan telecomm -site boston
-family PRODA -user bostonadmin -password secret -workdir
d:\shipping\temp -fship -sclass cq_default bangalore
```

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

   ```
   multiutil syncreplica -import -clan telecomm -site bangalore
   -family PRODA -user bangadmin -password secret -receive
   -sclass cq_default
   ```

4. At the new mastering replica, verify that mastership has been received:

   ```
   multiutil describe -clan telecomm -site bangalore -family
   PRODA -user bangadmin -password secret workspace:”Public
   Folders\Triage\project report”
   multiutil: The mastership of workspace:Public
   Folders\Triage\project report is ‘banaglore’
   ```

Transferring mastership of a user or group

This section describes how to change the mastership of a ClearQuest user or group using the **chmaster** command.

Only ClearQuest users with the Super User privileges can change the mastership of an object. In addition, mastership can only be changed at the site of the mastering replica.

Note: If your site receives a synchronization packet that contains user administration changes such as new users or groups, you must upgrade the user database at your site to use the changes.

Changing the mastership of a user or group involves the following steps:

1. Use the chmaster command at the mastering site to change the mastership of the user(s) or group(s) to the new site.
2. At the old mastering site, create and send an update packet to the new mastering site.
3. At the new mastering site, import the update packet that contains the mastership changes.
4. At the new mastering site, upgrade the user database at the site where the chmaster command was run to ensure that the new user information is propagated to the user database at the old mastering site.
5. At the new mastering site, send a synchronization packet to ensure that other replicas are updated with the change. No user database upgrade is necessary at other sites.

Understanding mastership and user administration

For more information about administering users in a ClearQuest MultiSite environment, see the *Rational ClearQuest Administrator's Guide*.

Although user administrators cannot modify users which are not mastered at their local site. User administrators at replica sites can add users, assign database privileges and create user groups.

Keep in mind that any changes in users or groups must be propogated through a combination of synchronization and user database upgrades.

In particular, if your site receives a synchronization packet that contains user administration changes such as new users or groups, you must upgrade the user database at your site to use the changes.

Adding a non-mastered user to locally-mastered group

You can add non-mastered users to a group as long as you have mastership of the group you want to modify. However, as with any user or group changes, a synchronization cycle as well as a user database upgrade at the receiving site must take place for the complete change to be in effect.

Specifying users and groups in the chmaster command

When changing mastership of users or groups, the entity-selector format is `user:<user_name>` for users and `group:<group_name>` for groups.

Use the following guidelines when denoting users and groups with the chmaster command:

- Use quotes if the user or group name includes spaces.
- In many cases, user and group names are case-sensitive, so follow the same case as was used when the user or group was created.

For example, to denote the user **John Smith**, you'll need to type the following:

```
user:"John Smith"
```

For example, to denote the group **managers**, you'll need to type the following:

```
group:managers
```

Example of changing mastership of a user

The following example changes the mastership of users John Smith and Jane Doe from the *boston* site to the *bangalore* site.

1. At the mastering replica (*boston*), enter a **chmaster** command to change the mastership of users John Smith and Jane Doe to the *bangalore* site:

   ```
   multiutil chmaster -clan telecomm -site boston -family PRODA
   -user bostonadmin -password secret bangalore user:”John
   Smith” user:”Jane Doe”
   multiutil: The mastership of record “Jane Doe” of type “user”
   has been changed to site ‘banaglore’
   multiutil:The mastership of some users or groups have been
   transferred from this site. The local user admin must update
   user databases at the new mastering site 'SANFRAN1_HUB'
   before these changes will be visible to any user database.
   ```

2. At the old mastering replica, export an update packet to the new mastering replica’s site:

   ```
   multiutil syncreplica -export -clan telecomm -site boston
   -family PRODA -user bostonadmin -password secret -workdir
   d:\shipping\temp -fship -sclass cq_default bangalore
   ```

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

   ```
   multiutil syncreplica -import -clan telecomm -site bangalore
   -family PRODA -user bangadmin -password secret -receive
   -sclass cq_default
   ```

4. At the new mastering replica (bangalore), upgrade the user database with the new user information. For more information about upgrading user databases, see the *Administering users* chapter in the *Administering Rational ClearQuest* manual. It can be found in PDF format in the \\ClearQuest\books directory.

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

   ```
   multiutil describe -clan telecomm -site bangalore
   -family PRODA -user bangadmin -password secret
   user:”John Smith” user:”Jane Doe”
   multiutil: The mastership of user:John Smith is ‘banaglore’
   multiutil: The mastership of user:Jane Doe is ‘banaglore’
   ```

Changing mastership of a working schema repository

This section describes how to change the mastership of a working schema repository using the **chmaster** command.

The working schema repository is responsible for modifying schemas and adding new replica families to a clan, see *Understanding the role of the working schema repository* on page 3. If you want to change the site where these tasks are done, you'll have to change mastership of the working schema repository.

To transfer the mastership of a working schema repository:

1. At the working schema repository replica, enter a **chmaster** command:
   ```
   multiutil chmaster -clan telecomm -site paris -family MASTR
   -user parisadmin secret boston -working master
   ```
2. At the old working schema repository, export an update packet to the new mastering replica:
   ```
   multiutil syncreplica -export -family MASTR -u parisadmin -p
   secret -workdir c:\temp\shipping -fship -scl cq_default
   ```
3. At the new working schema repository, import the packet:
   ```
   multiutil syncreplica -import -family MASTR -u bostonadmin -p
   secret -receive -scl cq_default
   ```

Forcing a transfer of mastership

Use the **–force** option with **chmaster** only when a replica is no longer available. Using **–force** to grab mastership from an available replica causes irreparable inconsistencies among the replicas in the replica family.

Fixing an accidental mastership change

If a mastership change is made in your replica by mistake, you'll need to wait for the next synchronization cycle and request that the new mastership replica change it back to your replica. Remember that until the next synchronization cycle, the mastership is "in the packet" and cannot be changed.

Follow these steps to undo the change:

1. At your replica, complete the transfer by sending an update packet to the new mastering replica.
2. At the new mastering replica, ask the ClearQuest MultiSite administrator to complete these steps:
   a. Import the packet.
   b. Change mastership back to your replica

c. Export an update packet to your replica.

3. At your replica, import the packet.

Appendixes

Troubleshooting MultiSite operations

A

This chapter describes common situations in which running a MultiSite command produces an unexpected result, sometimes accompanied by a warning or error message. The situations fall into these categories:

- **Expected conditions** occur because certain inconsistent changes at different replicas cannot be avoided. In many cases, a MultiSite operation resolves the inconsistency, so you need not take any action.
- **Recoverable errors** are user errors, hardware glitches, and other problems that you resolve by performing a recovery procedure.
- **Serious errors** are problems that may require assistance from Rational Technical Support.

The organization of the descriptions follows the general ClearQuest MultiSite data flow: from replica creation through the phases of replica synchronization—export, transport, and import. This chapter also describes replica restoration and replacement.

For information on changing mastership, see Chapter 6, *Managing mastership*.

Replica-creation problems

Problems with replica creation can occur during the export phase or the import phase.

Export phase

If the **mkreplica –export** command finds that a replica with the specified name exists in the database family (`Replica` *replica-name* `already exists`), select another name for the new replica, and reenter the **mkreplica –export** command.

If **mkreplica –export –fship** fails while it is transporting the packet, it does not remove the new replica's replica object at the creating site. To complete the replica creation, use **shipping_server** to transfer the replica-creation packet.

Import phase

If the **mkreplica –import** command fails for any reason (network outage, not enough disk space, etc.), you'll need to rerun the command. However it is possible that mkreplica -import will succeed with an import of the replicated schema repository but fail on the import of the user database replica. In that case, you'll only need to rerun the command specifying only the user database parameters.

For complete documenation on mkreplica, see *mkreplica* on page 162.

If **mkreplica -import** fails, you will see an error message detailing the failure.

If the import failed during the replicated schema repository import, do the following:

1. Delete the vendor databases intended for the replicated schema repository and user database replica.
2. Delete the newly created database name. The newly created database set name is in the following format: CQMS.<clan_name>.<site-name> where <*clan name*> is the name of the clan to which the replica belongs and <*site name*> is the name of your site.

   Enter the following to delete the database set name:
   ```
   installutil dropdbset CQMS.<clan_name>.<site_name>
   ```
3. Recreate the vendor databases.
4. Re-run the **mkreplica -import** command.

If the import failed during the user database replica import, but successfully imported the replicated schema repository, do the following:

1. Delete the vendor database(s) that were intended for the user database replica(s).
2. Re-create the vendor database(s).
3. Re-run the mkreplica import command, omitting the repository database options. For example:

   ```
   multiutil mkreplica -import -clan telecomm -site osaka
   -user cqjapanadmin -p secret -database cq_userdb
   -vendor SQL_SERVER -dbologin juseradmin secret -rwlogin
   juseradmin secret
   ```

Synchronization export problems

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

Cannot find oplog

syncreplica –export can fail with the following warning message:

```
Can not find oplog from replica replica-name with id oplog-ID
Gap in oplog entries may indicate missing oplog entries
```

(For more information on oplog entries, see *Database operations and the oplog on page 8* and *Scrubbing parameters for database replicas* on page 67.)

This error occurs when the sending replica's epoch number matrix does not match its set of oplog entries. For example:

- Before sending an update from **sydney** to **buenosaires**, **syncreplica** checks the epoch number matrix for **sydney**. It determines that the last **sydney** operation sent to **buenosaires** was 3620.
- **syncreplica** finds that oplog scrubbing in the **sydney** database has removed some of the operations that follow 3620. The earliest **sydney** operation remaining in the oplog is 5755.

This discrepancy may be an expected condition. For example, when a replica family changes its update topology, hosts that have not communicated with each other in the past start exchanging update packets. Synchronizing two replicas (**syncreplica –export** followed by **syncreplica –import**) updates epoch number matrix rows for the sending and receiving replicas, but it does not revise the row for any other replica. If two replicas rarely (or never) send updates to each other directly, the relevant rows in their epoch number matrices are quite out of date (possibly consisting of all zeros). This is not a problem, as long as the replicas receive each others' operations indirectly (for example, through a hub replica).

In this case, you must inform **sydney** about the true state of **buenosaires**, information that it has not received through the standard synchronization-update mechanism. This information enables **sydney** to determine which oplog entries must be sent to **buenosaires**.

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 following error messages are generated by the **mkorder**, **mkreplica**, **shipping_server**, and **syncreplica** commands.

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*.
cyclic delivery route detected to host *hostname* (via *next-hop-hostname*) for order *shipping-order-pname*	Sending the file to the *next-hop-hostname* specified in a ROUTE entry in the shipping.conf file or in the Routing Information section in the MultiSite Control Panel yields a circular delivery route.
file *file-pname* does not contain a valid shipping order	shipping_server attempted to process a file that is not a shipping order.
for security reasons, shipping order *shipping-order-pname* cannot be processed: data file *file-pname* must be in the same directory as the shipping order	A shipping order and its associated packet file must be in the same directory. This security feature prevents transmission of arbitrary files.
giving up trying to return order *shipping-order-pname* to host *hostname* (original data file was *file-pname*)	shipping_server cannot return a packet or other file to its original sending host (for example, because its shipping order expired) and has deleted the shipping order and data file.
ignoring shipping bay *storage-bay-pname*: *reason*	The storage bay directory specified in the shipping.conf file or MultiSite Control Panel is inaccessible, doesn’t exist, and so on.
shipping order *shipping-order-pname* not found (perhaps previously sent?)	During receipt handler processing, the shipping_server cannot find the shipping order of a packet that is to be forwarded to another host. A shipping_server –poll invocation may have sent the packet already. (If the packet is to be applied to replicas on the host, the imports occur before the packet is forwarded. This leaves a window of opportunity for a scheduled polling operation to send the packet.)

Invalid destination

The local host's **hosts** file, **hosts** NIS map, or Domain Name Service must list one of the following hosts:

- Destination host
- Next-hop host corresponding to the destination host (on UNIX, defined in a **ROUTE** entry in the host's **shipping.conf** file; on Windows, defined in the **Routing Information** section in the host's **MultiSite Control Panel**.)

Note: If hosts in your network are known only by their IP addresses, you can use the IP addresses instead of host names.

In the absence of such entries, **shipping_server** fails, because it cannot determine where to deliver the packet. In this case, it writes error messages to its log file.

If the destination host name was misspelled, use the **mkorder** command to create a new shipping order with the correct host name. If a host name is misspelled in a **mkreplica –export** command, the incorrect host name is recorded in the replica. Verify the error with **lsreplica –long**, and correct the spelling with **chreplica**.

In other cases, you may have to revise the host's database of remote hosts. The sending host must be able to communicate with the receiving hosts through TCP/IP channels. Use the **rcp** command on the sending host to copy a file to the receiving host. If it fails, you have a setup or networking problem with your host. If the command succeeds, contact Rational Technical Support.

Delivery fails

Each time **shipping_server** cannot deliver a packet to a valid destination host, it logs error messages:

- (On UNIX) In file /var/adm/atria/log/shipping_server_log and writes a message to the terminal device, if there is one.
- In the Windows event viewer. It writes log messages to file clearcase-home\var\log\shipping_server_log.

If the problem is temporary (remote host is down, network connections are down, and so on), a subsequent invocation of **shipping_server –poll** will transmit the packet successfully. If the problem is not temporary, the shipping order may expire eventually.

Shipping order expires

If the **shipping_server** finds that a shipping order has expired, it attempts to return the packet to the originating host. Also, it sends a mail message to one or more administrators on the original sending host, and sends another mail message when the packet is returned to the original sending host. On Windows, if e-mail notification is not enabled, **shipping_server** writes a message to the Windows event viewer and records the error in the clearcase-home\var\log\shipping_server_log file.

Use the **lspacket** command to check the *return bays* on your host. The packet files may have been returned by store-and-forward. If so, try again to deliver the packet:

- Fix the store-and-forward packet-delivery mechanism (for example, by fixing the network connection). Then, use **mkorder** to create a new shipping order for each physical packet file in the return bay.
- If you cannot fix the store-and-forward mechanism, deliver the packet by some other means. For example, copy the packet file to a diskette, and mail the diskette to the remote sites.

If the packet files are not in your host's return bays, they may be in transit. Search for the files immediately, because a packet that cannot be returned to its home host within 14 days is deleted.

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 at your site. These are the symptoms:

- One or more replicas at your site are not being updated on their regular schedules.
- A **lspacket** command shows unprocessed packets accumulating in the storage bay. These packets depend on the missing packet and cannot be processed.

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

1. Enter a **syncreplica –import –receive** command, which processes all incoming packets in the storage bay in the correct order. If **syncreplica** refuses to process any of them, a packet is missing.
2. Enter a **syncreplica –import** command that specifies the oldest packet in the storage bay:

```
multiutil syncreplica -import packet-pathname
Sync. packet packet-pathname was not applied to replica ...
 - packet depends on changes not yet received
Packet requires changes up to 872; replica has only 756 from
replica: sanfran_hub
Packet requires changes up to 605; replica has only 500 from
replica: bangalore
```

In this example, one or more update packets are missing, containing operations 757–872 originally occurring at replica **sanfran_hub** and operations 501-605 from **bangalore**. In general, a packet can contain operations from several replicas; the **syncreplica –import** command fails if operations are missing from *any* replica.

Locate the missing packets. They may be on a magnetic tape that you forgot to process or in packet files that were not processed because your store-and-forward configuration (the **shipping.conf** file on UNIX; the **MultiSite Control Panel** on Windows) specifies the wrong storage bay. If you locate the missing packets:

1. Process the missing packets by naming them in a **syncreplica –import** command. (Multiple packet files are imported in the correct order, regardless of the order of the command-line arguments.)
2. Process all the update packets that have accumulated in the storage bay by entering a single **syncreplica –import –receive** command.

If you cannot locate the missing packets, go to *Recovering from lost packets on page 94*.

Packet is not applicable to any local replicas

Import can fail with the following message:

```
multiutil: Error: Sync. packet pathname is not applicable to
any local replicas
```

This error can occur when a replica has been moved and the hostname property has not been updated with the **chreplica** command.

1. Verify that the hostname property is wrong, use the **lsreplica** command. For example, if the above error occurred at the bangalore replica:
   ```
   multiutil lsreplica -site bangalore -user
   bangaloreadmin -p secret -long bangalore
   Name:bangalore; Clan:TELECOMM; Family:PRODA; Host:shiphost1;
   Status: NORMAL; Description:Production database
   ```
2. If the hostname is incorrect, use the **chreplica** command to change it. For this example, run **chreplica** at the **bangalore** replica:
   ```
   multiutil chreplica -site bangalore -user bangaloreadmin -p
   secret -host shiphost2 bangalore
   Updated replica information for "bangalore".
   ```
3. Send an update packet to the other replicas in the family.

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.
- ClearQuest or MultiSite licensing failure.
- Multiple imports occurring simultaneously.

Make sure that multiple **syncreplica –import** commands do not run in the same replica simultaneously.

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 a magnetic tape or diskette 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 92.)

Lost replica-creation packet

To recover a lost replica-creation packet:

1. At the replica where the **mkreplica –export** command was entered, remove the new replica with **rmreplica**.
2. Reenter the **mkreplica** command.

Lost update packet

The **syncreplica** –export command assumes successful delivery of the update packet it generates. For example, when replica **lex_hub** sends an update to replica **sanfran_hub**, the **syncreplica** command assumes that the operations originating at lex_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 database family.

However, if the packet is lost, this assumption is invalid, and **lex_hub** must reset its estimate of the state of replica **sanfran_hub**. After this correction is made, the next update packet sent from **lex_hub** to sanfran_hub contains the operations **sanfran_hub** needs.

To reset the epoch row, use the method described here.

1. At the receiving site (sanfran_hub), use the **lsepoch** command to display the replica's epoch number matrix:

```
multiutil lsepoch -clan telecomm -site sanfran_hub -family
PRODA -user sfadmin -p secret sanfran_hub
Multiutil: Estimates of the epochs from each site replayed at
site 'sanfran_hub' (@goldengate):
BANGALORE: 950
LEX_HUB: 1300
SANFRAN_HUB: 2000
```

2. At the sending site (lex_hub), use this output in a **chepoch** command:

```
multiutil chepoch -clan telecomm -site lex_hub -family PRODA
-user lexadmin -password secret bangalore=950 lex_hub=1300
sanfran_hub=2000
Change oplog ID in row "sanfran_hub", column "bangalore" to
950 [no] yes
Change oplog ID in row "sanfran_hub", column "lex_hub" to
1300 [no] yes
Change oplog ID in row "sanfran_hub", column "sanfran_hub" to
2000 [no] yes
Epoch row successfully set.
```

Resolving naming conflicts

If your site has not adopted a naming convention for stateless record types users, groups or Workspace items, it is possible to have naming conflicts with these items.

When naming conflicts are found in stateless record types during synchronization, ClearQuest MultiSite internally modifies the name of the stateless record to be unique.

ClearQuest MultiSite does this by using the ratl_keysite field. The ratl_keysite field is part of the unique key of a stateless record type and is null until a naming conflict occurs. When a naming conflict is detected, the ratl_keysite field contains the name of the originating replica.

The ratl_keysite field is a system field that is a reference to the replica record for the site where the record originated. Once this field is populated it cannot be changed.

For example, a new user named **jsmith** is created at two replicas during the same time period between synchronizations. When each site synchronizes, there will seemingly be two user records with identical names. To maintain data integrity during the synchronization, ClearQuest MultiSite automatically completes the ratl_keysite field. You should rename the conflicting records as soon as possible.

Although you cannot see such changes taking place during synchronization, ClearQuest MultiSite provides several ways to detect changes to the unique key when accessing stateless records.

- Create a query designed to detect naming conflicts and then manually rename conflicting records.
- At the working schema repository site, edit the schema used for a replica family to include a field hook designed to detect naming conflicts and rename them.
- At the working schema repository site, add the ratl_keysite field to the form of each stateless record types.

Creating a query to detect record naming conflicts

You can use the ratl_keysite field in queries designed to find stateless records whose unique keys have been modified.

1. Use the ratl_keysite field as both a Display field and Filter when creating a query on the respective stateless record type.
2. Define the ratl_keysite filter to look for all records whose ratl_keysite field is not null.
3. If the query finds any duplicately named records, rename them using a site identifier that is accessible to users. For example, rename jsmith to jsmith_sitea.
    - For user records, use the User Administration menu from ClearQuest Designer. You must have User Administrator privileges.

Warning: ClearQuest does not allow you to rename user groups, see the ClearQuest MultiSite Release Notes for more information.

Modifying the schema to detect record naming conflicts

You can use the ratl_keysite field in your schema to assist you in viewing/modifying records with naming conflicts in the multiple ways.

- Add the ratl_keysite field to the form of any stateless record type where you expect naming conflicts to arise.
- Add a Value Changed field hook that either renames the record automatically or gives a message telling you that the record needs to be renamed.

Restoring replicas

Occasionally, a replica 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. If an unreplicated database 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 a replica, 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 family before development can proceed at your site. Failure to reestablish consistency can lead to irreparable damage.

Because this procedure involves substantial effort, it is intended for situations where serious damage has occurred. (For example, the disk containing a replica is unusable.)

Restoring a replica from backup

To restore a replica from backup:

1. Use your vendor database tools to restore a copy of the replicated database from backup.
2. Use the **restorereplica** command to start the restoration procedure. You must have Super User privileges.

   ```
   multitutil restorereplica [-cl·an clan-name] [-site
   site-name] [-fam·ily family-name]
   -u·ser username -p·assword password [-force] [-completed]
   [-replace][replica...]
   ```

   This places a special lock on the replica. Between this point and the completion of Step 6, the **syncreplica –import** command adjusts the ClearQuest 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. 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.
5. Wait for each other replica in the family to send an update packet to the restored replica. As in Step 4, 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.

6 Process the incoming update packets with **syncreplica –import**. When your replica has received return acknowledgments from all other replicas in its family, **syncreplica –import** reports that restoration of the replica is complete:

```
Database <name> is unlocked after restoration.
```

Development work in the replica can now resume.

Cleaning up from accidental deletion of a replica

This situation is a more catastrophic variation of the problem described in *Restoring a replica from backup* on page 98: a replica is lost, and there is no backup to be restored. The procedure for handling this situation is similar to that in *Removing a replica* on page 64.

You can perform this procedure at any replica within the same family as the deleted replica.

1 Transfer mastership of all the objects that were mastered by the deleted replica. For example, if replica **tokyo** is deleted, enter this command at replica *sanfrancisco*:

```
multiutil chmaster -family PRODA -user sfadmin -p secret
sanfrancisco -all -force tokyo -long sanfran_hub
```

Warning: Incorrect use of **–all –force** can lead to irreparable inconsistencies among the replicas in a family.

2 From the sanfrancisco site, remove the deleted replica.

```
multiutil rmreplica -family PRODA -user sfadmin -p secret
tokyo
```

3 Send an update packet to all other replicas in the family, to inform them of the mastership changes and the replica deletion.

```
multiutil syncreplica -export -clan telecomm -site
sanfrancisco -family PRODA -user sfadmin -p secret -workdir
c:\temp\shipping -fship -sclass cq_default bangalore paris
boston
```

Configuring store-and-forward

B

ClearQuest MultiSite allows you to use transport replica creation and sychronization packets several ways. This chapter covers the following topics:

- Using Store-and-Forward
- Configuring Store-and-Forward
- Automating synchronization
- Using ClearQuest MultiSite through a firewall
- Installing Store-and-Forward on a firewall host

Using Store-and-Forward

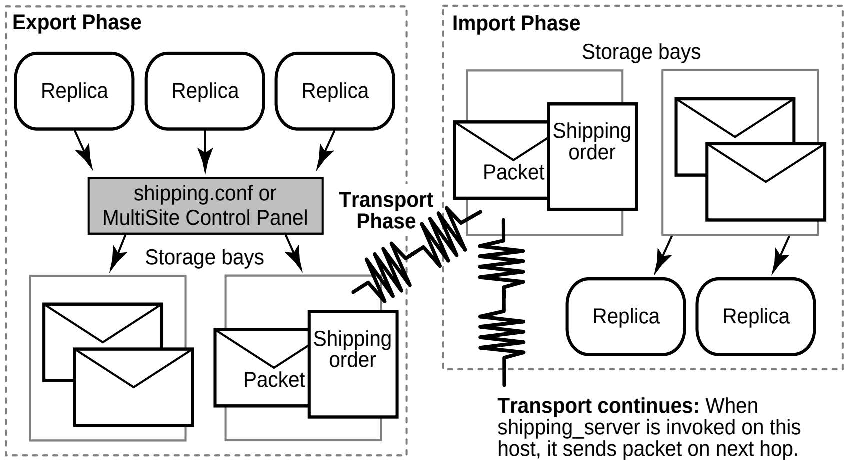
The ClearQuest MultiSite *store-and-forward* facility is a file-transfer service that automates the *transport phase*. This facility can handle packets of any size, can route files through a series of ClearQuest MultiSite hosts, one hop at a time, and includes support for handling data-communications failures.

The major components of the store-and-forward facility include:

- MultiSite Control Panel (Windows) or shipping.conf (UNIX)
- Rational Shipping Server

The store-and-forward process is illustrated in below and described in the following sections.

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



Packet storage during the export and import phases

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

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

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

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

Packet transport

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

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

Submitting packets to Store-and-Forward

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

- The **–fship** option places the packet files and shipping order files in one of the host's storage bays, and runs **shipping_server** to send the packet files to their destination host or route them to an intermediate host.
- The **–ship** option places the packet files and shipping order files in a storage bay, but does not invoke **shipping_server**. The packet files are sent the next time the **shipping_server** polls the appropriate bay.

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 **MultiSite Control Panel** (Windows).

For example:

- 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**, **lex-hub**, and **tokyo** in the **Destination Hostnames** box.

Any packet whose final destination is host **sanfran-hub**, **lex-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 **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 Rational Shipping Sever 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 successfully (for example, because the receiving host is unavailable), **shipping_server** generates an error message and log file entry, then exits. Administrators can set up a retry scheme to control its frequency:

- After successful transmission of a packet to another site, **shipping_server** deletes both the packet and its shipping order. After a transmission failure, a packet and its 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**.

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

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

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

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

- It modifies the shipping order to return the packet to the original sending host, where it is placed in a special *return bay.*
- It sends an electronic mail message to one or more addresses on the original sending host. (Another mail 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 104.* 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.

Sending files that are not packets

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

- To send a file immediately, use the **–fship** option with **mkorder.** This example sends a file to the boston and bangalore hosts:
  ```
  mkorder -data e:\shipping\packet1.xml -fship boston bangalore
  ```
- To store the file in a shipping bay so that **shipping_server** will send the file the next time it runs, use the **–ship** option:
  ```
  mkorder -data e:\shippping\packet1.xml -ship -copy
  ```

Warning: 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.

Configuring Store-and-Forward

Before using store-and-forward, you must make sure you have the appropriate disk space, configure the MultiSite Control Panel or shipping.conf to create storage classes for packets.

ClearCase MultiSite considerations

The store-and-forward facility and the Rational Shipping Server are used for both ClearQuest MultiSite and ClearCase MultiSite. If you are using the same shipping server host to transport packets for both ClearQuest MultiSite and ClearCase MultiSite, you'll need to consider the following:

- Installation of the Rational Shipping Server may vary if you are trying to install ClearQuest MultiSite or the Rational Shipping Server on a machine that has ClearCase or ClearCase MultiSite installed, see *Installing Rational ClearQuest* for more information.
- You cannot use the same storage class for ClearQuest MultiSite and ClearCase MultiSite. The name of the default storage class for ClearQuest MultiSite is *cq_default*. The name of the default storage class for ClearCase MultiSite is *default*.You must use different storage classes with unique names, see *Differentiating packets with storage classes* on page 109.

Disk space needed for ClearQuest MultiSite

Regardless of which transport mechanism you use, you will need to set up ClearQuest MultiSite storage bays to store and receive update packets from replicated databases. You will need to allocate space for these directories on a machine. The amount of disk space varies according to the size of the update packets you will be sending.

If you are using the Rational Shipping Server to transport updates, the machine on which the storage bays reside needs to be accessible to the replicas that will be updated and any replicas that will be generating ClearQuest MultiSite packets.

The storage bays hold ClearQuest MultiSite packets, along with their corresponding *shipping order* files. The following table describes the amount of available disk space needed on the disk partition where the storage bay is located.

Type of packet	Disk space needed
replica-creation	At least twice the size of database and schema repository. Packets can be up to four times as large as the databases they are created from.
update	On Windows, twice the size of the largest packet to be stored in bay. The reason is that there may be two instances of the same packet in the bay at one time: one on its way to another destination, and another waiting to be applied to the replica on the current host.

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

Setting up the MultiSite Control Panel or shipping.conf

ClearQuest MultiSite allows you to customize the settings of the store-and-forward utility using the MultiSite Control Panel (Windows) or the shipping.conf file (UNIX).

You must configure the store-and-forward utility before creating or receiving any replica packets. The MultiSite Control Panel (Windows) or the shipping.conf file (UNIX) controls the operation of the store-and-forward facility on the local shipping server host machine. It manages the locations of storage and return bays that store and receive update packets to and from other replicas.

To view the MultiSite Control Panel:

1. Click **Start > Settings > Control Panel**.
2. Double-click the MultiSite icon.

For more information on specifying settings, see the *MultiSite Control Panel* on page 178.

For UNIX, you'll need to edit the shipping.conf file. For more information, see *shipping.conf* on page 197.

The settings for the store-and-forward facility are host-specific. Each store-and-forward host can also support multiple storage classes. For each storage class, you can specify locations of storage and return bays, routing information to support multihop packet delivery, specifications to handle failure-to-deliver situations, receipt handlers, and so on.

For more information on specifying settings, see *MultiSite Control Panel* on page 178 or *shipping.conf* on page 197.

Differentiating packets with storage classes

You can configure the store-and-forward facility to handle updates for different replica families 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: The default storage class used by ClearQuest MultiSite varies according to command. All ClearQuest MultiSite commands that use the -sclass argument use the default storage class name of *cq_defaul*t, except **mkorder** and **shipping_server**, which default to the storage class name of *default*.

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

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

Creating a storage class

There are a few guidelines you should use when creating a storage class.

- The default storage class used by ClearQuest MultiSite varies according to command. All ClearQuest MultiSite commands that use the -sclass argument use the default storage class name of *cq_defaul*t, except **mkorder** and **shipping_server**, which default to the storage class name of *default*.

- The storage bay should be unique. You cannot use the same name or directory as you do for ClearCase MultiSite packets.
- The directory you specify should be on a drive that has sufficient room for the packet size you specify.
- Although you specify the administrator's e-mail address, you'll need to use the *control_panel* on page 139 to complete the e-mail notification process.

Receiving error notification

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 was 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 on enabling e-mail notification, see *MultiSite Control Panel* on page 178 and *control_panel* on page 139.

Automating synchronization

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

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

You can use a third-party scheduling tool in conjunction with ClearQuest MultiSite receipt handlers (see *MultiSite Control Panel* on page 178 and *shipping.conf* on page 197 for more information about receipt handlers) to automate the exporting and import process.

You can define receipt handlers in the MultiSite Control Panel of the shiping.conf (UNIX) for different shipping classes.

A sample scheduling script (receipt handler)

The following Perl script can be used as a starting point to setting up an automated synchronization of replicas. Before you can use this script, you will need to modify it by doing the following:

1. Change the hard-coded value for your MultiSite working directory
2. Change the hard-coded values that define the MultiSite clans, families, and sites that should be known to this program
3. Change the hard-coded variables to the MultiSite user administration logon and password.

Once you have updated the script and tested it thoroughly, you may then configure an external scheduling program, such as the Windows "Scheduled Tasks" facility, to automate its execution.

```
#
# MSAuto.pl -- MultiSite Automation
#
# This script automates routine MultiSite tasks, by issuing
# the appropriate MultiSite commands to synchronize the
# replicas.
# This program currently takes the approach of always using
# the '-fship' option when sending updates (to tell the
# shipping server to send them immediately). Another approach
# would be to use 'syncreplica -ship' (to stage the updates
# in the output shipping bay) and then to periodically use
# 'mkorder -fship' to tell the shipping server to actually
#ship the packets.
#
# This program operates in one of two modes:
# 1. Sync now -- The program will receive any pending
# updates, then send any pending updates, then exit. Use this
# mode if you want to run the synchronization immediately or
# if you want to schedule the program's execution via some
# external scheduler package, such as the Windows "Scheduled
# Tasks" facility or the ClearCase scheduler.
# 2. Loop and wait -- The program receives, sends, then
# sleeps a specified number of seconds, then loops back and
# receives, sends, and sleeps again. Use this mode if you
# want the program to essentially act as its own scheduler.
#
# The choice of mode is determined by a command-line operand:
# MSAuto.pl [MaxLoops [TimeToWait]]
#
# The default is "Sync now" mode, where the program performs
# the synchronization commands once each and then exits. If
# 'MaxLoops' is specified but TimeToWait is not, then the
# default is 30 seconds between iterations.
#
# This program writes out messages indicating what's going
# on, and the messages go both to STDOUT (normally a command
# prompt console window) *and* to a 'log' file. The log file
# is, by default, located in the CQ# program directory and it
# has the same name as this program with a suffix of ".log"
# appended. Each execution appends new lines to the end of
# the file.
#
# Set the debug level:
# 0    = Do real work; don't produce any debugging output
# 1..9 = Show diagnostic info and actually do work too
# (higher numbers show more granular output)
# 10+  = Show diagnostic info, don't actually do real work
#
$DEBUG = 0;
$MeVersion = "0.7";  # This program's version
$MyWorkDir = "C:\\TEMP\\MS"; # Work directory...
# YOU WILL NEED TO EDIT THIS SECTION
# Define the MultiSite clans, families, and sites
# Define the AdminUser and AdminPassword
#
# Syntax: $MSClan{'CLAN'} = "AdminUser AdminPassword
FAMILY\@SITE1[,SITE2...][;FAMILY\@SITE1[,SITE2...]...";
$MSClan{'RAMBU'} = "admin cqadmin
RAMBU\@RAMBUCUP,RAMBUREP1;RTEST\@RAMBUCUP,RAMBUREP1";
####$MSClan{'RAMBU'} = "admin cqadmin
RAMBU\@RAMBUCUP,RAMBUREP1";
# Build the commands needed to sync this site with other
# sites...
sub MkMSCommands() {
    &MsgOut(2, "MkMSCommands() starting...\n");
# First, query the registry to get a list of the MultiSite
# "clans"
# (schema repo's) we are a member of, and our "site" name
# within each
# clan...
    use Win32::TieRegistry;
    $CQDBRegKey = "HKEY_LOCAL_MACHINE\\SOFTWARE\\Rational
Software\\ClearQuest\\2001A.04.00\\Core\\Databases";
    if (! ($DBSetList = $Registry->{$CQDBRegKey})) {
        &MsgOut(0, "ERROR: Can't read registry key
'$CQDBRegKey'\n"
                   . "Unable to determine MultiSite clan
membership. Exiting.\n");
        &AllDone(1);
        }
    foreach ( keys (%$DBSetList) ) {
        if (/^CQMS\.(\w+)\.(\w+)\\$/) {
              &MsgOut(3, "This machine is a member of clan '$1'
with site ID '$2'.\n");
            $MyClanList{$1} = $2;
        }
    }

# For each clan (schema repository) we're a member of, build
# the commands
# necessary to receive its updates and to send updates to
# each of the sites containing any replicas of any of its
# families
# (databases)...
    my($CMDs) = "";
    foreach $MyClan (sort keys %MyClanList) {
        $MySite = $MyClanList{$MyClan};

# Get all the info for this clan...
        my($CQAdminUser, $CQAdminPswd, $ClanInfo) = split ("
", $MSClan{$MyClan});
        &MsgOut(3, "MyClan=$MyClan MySite=$MySite
CQAdminUser=$CQAdminUser " .
                   "CQAdminPswd=$CQAdminPswd
ClanInfo=$ClanInfo\n");

        foreach $FamilyInfo (split(";", $ClanInfo)) {
            &MsgOut(3, "FamilyInfo=$FamilyInfo\n");
            ($Family, $Sites) = split('\@', $FamilyInfo);
# First, see if our site is a member (don't try to
# synchronize if not!)...
            my($IAmInFamily) = 0;
            my(@OtherSites) = ();
            foreach $Site (split (",", $Sites)) {
                if ( (uc $Site) eq (uc $MySite)) {
                    $IAmInFamily = 1;
                } else {
                    push (@OtherSites, $Site);
                }
            }
            if ($IAmInFamily) {
                &MsgOut(3, "MySite=$MySite.
OtherSites=@OtherSites.");
# Build the "syncreplica -import" command to receive updates
# from all sites...
                $CMDs .= ("multiutil syncreplica -imp"
                      . " -u $CQAdminUser"
                      . " -p $CQAdminPswd"
                      . " -clan $MyClan"
                      . " -family $Family"
                      . " -receive\n"
                      );
# Build the 'syncreplica -export' command to send updates
# to all sites...
                $CMDs .= ("multiutil syncreplica -exp"
                      . " -u $CQAdminUser"
                      . " -p $CQAdminPswd"
                      . " -clan $MyClan"
                      . " -fam $Family"
                      . " -site $MySite"
                      . " -fship -wor $MyWorkDir -sc
cq_default"
                      . " @OtherSites\n"
                      );
                 }
        }
    }
    &MsgOut(2, "MkMSCommands() done.\n");
    $CMDs;
}
sub MkTimeStamp {
    local ($sec,$min,$hr,$mday,$mon,$yr,$wday,$yday,$isdst);
    ($sec,$min,$hr,$mday,$mon,$yr,$wday,$yday,$isdst) =
localtime(time);
# Note: You have to add '1900' to the year and '1' to the
# month to get
# human-recognizable results (see localtime() description for
# details).
    return sprintf
("%04d%02d%02d%02d%02d",$yr+1900,$mon+1,$mday,$hr,$min);
}
sub MkMsgHeader() {
    return (&MkTimeStamp() . " " . $Me . "[" . $$ . "] ");
}
sub MsgOut() {
    local($DebugLevel, $Msg) = @_;
    if ($DebugLevel <= $DEBUG) {
        foreach $ML ((split('\n', $Msg))) {
            my($M) = &MkMsgHeader() . $ML . "\n";
            print $M;
# If we've figured out where the LOG file is, open it, write
#this line, and close. If we haven't yet figured that out,
# queue this line so it'll be written eventually...
            if ($MyLogFile ne "") {
# Yes, we know where the log file is. Open it, send any
# pending messages, then this message, then clear the
# pending messages.
                open (LOG, ">>$MyLogFile");
                print LOG $PendingLogMessages . $M;
                $PendingLogMessages = "";
                close LOG;
            } else {
                $PendingLogMessages .= $M;
            }
        }
    }
}
sub DOsystem() {
    my($CMD) = shift;
    &MsgOut(1, "DOSystem($CMD)...\n");
    `$CMD` if ($DEBUG <= 9);
}
sub AllDone() {
    my($RC) = shift;
    &MsgOut(0, "$Me [PID=$$]: Done. Exit code=$RC.\n");
    if ($DEBUG > 0 || $RC > 0) {
        &MsgOut(0, "Press any key to exit...");
        system("pause >nul");
        &MsgOut(0, "\n");
    }
    exit $RC;
}
# Remove a directory including any subordinate files or
# subdirectories...
sub RmDir() {
    my($D) = shift;
    &MsgOut(1, "RmDir($D) starting...\n");
    if (-f $D) {
        &MsgOut(2, "$D is a file... deleting...\n");
        unlink $D if ($DEBUG < 10);
    } elsif (-d $D) {
        &MsgOut(2, "$D is a directory... calling RmDir to
remove files in it...\n");
        foreach (<$D\\*>) {
            &RmDir($_);
        }
       &MsgOut(2, "Back from removing files from $D. Removing
it...\n");
        rmdir $D if ($DEBUG < 10);
    }
    &MsgOut(1, "RmDir($D) done.\n");
}
# Get our program name...
@MePath = split(/\\/, $0);
$Me = pop(@MePath);
# Process command-line arguments (loop control)...
($MaxLoops, $TimeToWait) = @ARGV;
```

```
# Apply defaults, if necessary...
$MaxLoops = 1 if (! $MaxLoops);
$TimeToWait = 30 if (! $TimeToWait);
# Print 'greeting' info...
&MsgOut(0,"$Me [PID=$$]: MultiSite Automation Script version
$MeVersion starting...\n");
if ($MaxLoops > 1) {
    &MsgOut(0,"Program will loop $MaxLoops times, waiting
$TimeToWait seconds between iterations.\n");
}
&MsgOut(1,"DEBUG Level        = $DEBUG\n");
# Find out where CQ is installed...
$CQSubPath = ":\\Program Files\\Rational\\ClearQuest";
foreach ("C", "D", "E") {
    $CQPath = $_.$CQSubPath;
    $MUEXE = $CQPath."\\MultiUtil.exe";
    &MsgOut (2, "Checking for '".$MUEXE."'.\n");
    last if (-f $MUEXE);
}
if (! -f $MUEXE) {
    &MsgOut(0, "ERROR: Unable to find ClearQuest directory.
Quitting...\n");
    &AllDone(1);
}
&MsgOut(1, "ClearQuest Path    = $CQPath\n");
# Put the CQ directory onto the front of the path so it's
# easier to find MultiUtil.exe...
$ENV{'PATH'} = "$CQPath" . ";" . $ENV{'PATH'};
# Build the path to the LOG file...
$MyLogFile = $CQPath . "\\" . $Me . ".log";
&MsgOut(1, "Log file           = $MyLogFile\n");
# Gather info about the clans we're a member of, and using
# that information build the
# commands that will be used to actually synchronize us with
# all the sites we're connected to...
$MS_Cmds = &MkMSCommands();
&MsgOut(1, "Commands to synch us with other sites =
'$MS_Cmds'\n");
# Loop doing work until told to stop...
for ($i = 1; $i <= $MaxLoops; $i++) {
    &MsgOut(0, "Loop iteration $i of $MaxLoops...\n") if
($MaxLoops > 1);
# Loop through the list of commands...
    foreach $CMD (split ("\n", $MS_Cmds)) {
        &MsgOut(0, "COMMAND: $CMD\n");
        $Result = &DOsystem($CMD);
        chomp($Result);
        if ($Result eq "") {
           &MsgOut(0, "Command completed. Nothing to do.\n");
        } else {
            &MsgOut(0, "Command completed. Output was:\n" .
                  "$Result\n" .
                  "End of command output\n");
# Confirm the delete of packet file. If found, go out and
# actually delete the file...
            foreach (split ("\n", $Result)) {
                if (/^multiutil: Deleting packet (.*)$/) {
                    $F = $1;
                    &MsgOut(0, "Deleting '$F'...\n");
                    $NDeleted = unlink $F;
                    if ($NDeleted != 1) {
                        &MsgOut(0, "ERROR: Unable to delete
$F.\n  Error=$!.\nContinuing...\n");
                    }
                }
            }
        }
# Confirm the workdir was deleted. If the workdir still
# exists, delete it...
        if (-d $MyWorkDir) {
           &MsgOut(0, "$MyWorkDir not cleaned up by command.
Deleting...\n");
            &RmDir($MyWorkDir);
        }
    }
# Sleep the prescribed number of seconds before looping
# back...
    if ($i < $MaxLoops) {
        &MsgOut(0, "Sleeping $TimeToWait seconds...\n");
        sleep $TimeToWait;
    }
}
&AllDone(0);
```

Using ClearQuest MultiSite through a firewall

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

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

- Use an existing electronic mail mechanism as the transport.
- Use the standard **ftp** utility to transport packets.
- Use a custom TCP application.
- Install the Store-and-Forward software on a host configured to communicate through the firewall.

Note: When you use a file-based method for transport, you may need to use the **–maxsize** option to ensure the file is a manageable size.

Using e-mail

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

Advantages:

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

Disadvantages:

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

Automating the e-mail process

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

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

ClearQuest 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 on UNIX.

For security, you must encrypt the packets.

Using FTP

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

Advantages:

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

Disadvantages:

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

Using custom software

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

Advantages:

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

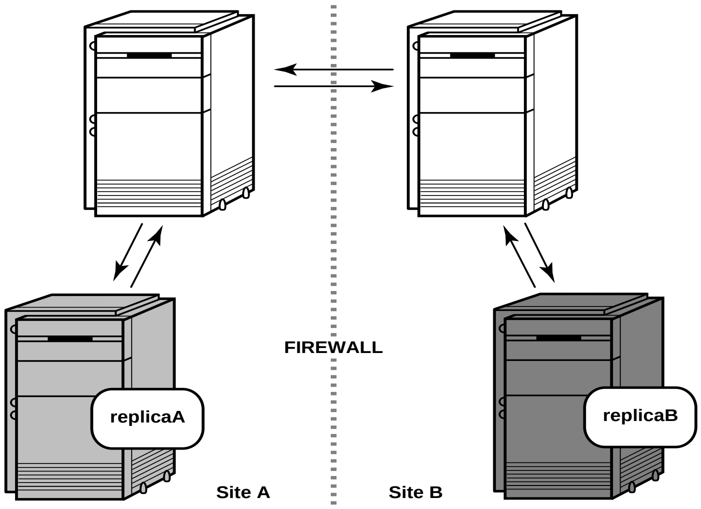
Disadvantages:

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

Installing Store-and-Forward on a firewall host

Note: Because of security concerns, we recommend that you use this method only if other methods are unsuitable for your site.

An alternative to using mail, **ftp**, or custom software is to install the store-and-forward software on a "firewall host," a host that can communicate through the firewall. ClearQuest MultiSite synchronization commands can forward data intended for systems on the other side of the firewall to this host. The software on this host then forwards packets through the firewall to the next hop. To specify the range of port numbers to be used on the host, you can use the environment variables **CLEARCASE_MIN_PORT** and **CLEARCASE_MAX_PORT**. In the figure below, the hosts that communicate through the firewall are the firewall hosts; they have the MultiSite Control Panel software installed on them, but not ClearQuest software. The replica server hosts have ClearQuest MultiSite installed on them.



Firewall issues

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

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

- Shipping bays can be overfilled.

  Using **shipping_server** on a firewall host enables anyone coming in from the network to fill shipping bays anywhere on the local network, on any machine where a **shipping_server** is available.

  To avoid full disks and the related problems:

  Ensure that all shipping bays in the local network are on partitions of their own, so that filling the bays do not degrade system performance.

  Install **shipping_server** only on machines that need it: machines used by ClearQuest MultiSite administrators.

- Packets are susceptible to snooping.

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

- Other servers can be accessible.

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

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

Installing shipping_server on a Firewall

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

For Windows installation instructions, contact Rational Technical Support.

Controlling ports used by albd_server and shipping_server

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

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

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

Guidelines for setting port values

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

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

Specifying port values on UNIX

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

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

1. On the firewall host, edit the file *ccase-home-dir*/**etc/atria_start**.
2. 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
```

Specifying port values

To specify minimum and maximum port values:

1. On the firewall host, open Control Panel and click the **System** icon.
2. Create two system environment variables, **CLEARCASE_MIN_PORT** and **CLEARCASE_MAX_PORT**, and specify their values.

MultiSite Command Line Reference

C

Command summary

Task	Command
Creating replicas	activate, mkreplica
Synchronizing replicas	syncreplica, lsepoch, chepoch
Changing mastership	chmaster, describe
Manage update packets	recoverpacket, mkorder, lspacket
Maintaining replicas	chreplica, lsreplica, restorereplica, rmreplica
Troubleshooting	lsepoch, chepoch, scruboplog, dumpoplog
Store-and-Forward	control_panel, shipping.conf, shipping_server

activate

Prepares a database set to be used by ClearQuest MultiSite.

Synopsis

activate [**–dbset** *dbset-name*] **–u·ser** *user-name* [**–p·assword**] *password*
–cl·an *clan-name* **–site** *site-name* **–host** *host-name*

Description

The **activate** command prepares a database set to be replicated. When you activate a database set, you give it a clan name and site name.

Before you can activate a database set, all user databases associated with that database set (schema repository) must be upgraded to the same version of ClearQuest.

When a database is activate, its logical name is changed to CQMS.<clan-name>.<site-name>. Once you activate a database, it can be accessed by additional multiutil commands, see *Using multiutil* on page 18

After you have activated a database set, you can replicated one or more user databases within that database set. To replicate a database that has been activated, use the **mkreplica** command.

Note: You only need to activate a database set once, before creating the first replica.

Options and Arguments

The following sections describe the options and arguments for use with **activate**.

–dbset *dbset-name*

The name of the database set you want to activate. You can omit this argument if your ClearQuest installation has only one database set (schema repository).

–u·ser *user-name*

The name of a user with ClearQuest Super User privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. You must have Super User privileges to administer ClearQuest MultiSite. You can omit this argument if the password is null.

–cl·an *clan-name*

Name by which the database set will be identified in subsequent multiutil commands. When you activate a database set, you also provide a clan name. The clan name must be unique and can be up to 50 characters long. The clan name cannot be changed after a database set has been activated.

–site *site-name*

Name by which the replica will be identifed in multiutil commands. The site name must be an identifier and can be up to 50 characters long.

–host *host-name*

The host name refers to the <machine name> where the storage bays for the respective replica are located and must be consistent with the name of the host specified when you configured the MultiSite Control Panel. The Rational Shipping Server must also be installed on the host machine.

Examples The following example activates the default database set, names the clan "telecomm" and names the site "boston". This replica will use server1 as its host.

```
activate -user bostonadmin -password secret -clan
telecomm -site boston -host server1
```

See also mkreplica

chepoch

Changes epoch number estimates.

Synopsis

chepoch [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
-u·ser *username* [**–p·assword**] *password*
[**–f·orce**] *replica* [*replica=value* ...]

Description

This command changes, in one replica, one or more of the *epoch numbers* that represent the replica's estimate of the epoch levels of other replicas. You cannot change a replica's own epoch numbers because these epoch numbers record the actual state of the replica.

You use this command if you need to resend a packet that did not get applied.

For more information about epoch numbers, see *Database operations and the oplog on page 8*.

Options and Arguments

–cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repositiory form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR.

Default: None.

–u·ser *user*

The name of a user with Super User privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. You must have Super User privileges to administer ClearQuest MultiSite.

–f·orce

Suppresses confirmation steps.

replica

Specifies the replica whose epoch row is to be changed; that is, changes the current replica's estimate of the state of *replica*. Use the site name to specify the replica.

[replica=value]

One or more arguments,

where

replica	Column of the epoch number matrix. This argument, along with the preceding *replica* argument, specifies a particular location in the matrix.
value	New epoch number to be entered at the specified matrix location.

Default: **chepoch** reads a set of *replica=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.

Examples

- Changes two columns of epoch estimates in the current replica's (*boston*) row for the *bangalore* replica.

```
multiutil chepoch -clan telecomm -site boston -family
PRODA -user bostonadmin -password secret bangalore
paris=500 sanfrancisco=750
```

```
Change oplog ID in "bangalore", "paris" to 500 [no] yes

Change oplog ID in "bangalore", "sanfrancisco" to 500
[no] yes
```

- Make the same change as in the proceeding example, but skip the confirmation step.

  ```
  multiuitl chepoch -clan telecomm -site boston -family
  PRODA -user bostonadmin -password secret -force
  bangalore paris=500 sanfrancisco=750
  ```

  ```
  Epoch row successfully set.
  ```

See Also

- lsepoch
- recoverpacket
- restorereplica

chmaster

Transfers mastership of a ClearQuest object from one site to another.

Synopsis

chmaster [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
  -u·ser *username* [**–p·assword**] *password new-master-replica*
  { { *entity-selector...* |{ **–all** [**–l·ong**] | **-workingmaster** }
  [**–force** *old-master-replica*]}
  }

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.

Only the replica that masters an object can transfer that object, unless you use the **-force** argument.

Mastership restricts some of the operations you can perform on an object. For more information about mastership, see *Managing mastership* on page 69.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repositiory form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR. You should specify the MASTR family when you are changing the mastership of the working schema respository with the **-workingmaster** option.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

new-master-replica

The name of the replica to which you are transferring mastership.

Default: None.

Specifying objects of which to change mastership

You can specify a particular object (*entity-selector*), that all objects for a particular family be transferred (**-all**) or that only the working master site be changed.

entity-selector

This is used to indicate the object type of which you want to change mastership. The following ClearQuest objects use mastership.

Change mastership Object	Syntax
Record	<record_type>:<record_id>
Users and groups	user:<login_name> or group:<group_name>
Workspace item	workspace:<query_name>
Stateless record which has a <site_name> See *Resolving naming conflicts* on page 95	<record_type>:<record_id>.<sitename>
Workspace item which has a value in the <site_name> field See *Resolving naming conflicts* on page 95	workspace:<query_name>.<site_name>
User or group which has a value in the <site_name field. See *Resolving naming conflicts* on page 95	user:<login_name>.<site_name> or group:<group_name>.<site_name>

Default: None.

–a·ll [–l·ong]

Transfers to *new-master-replica* mastership of all objects that are located in and mastered by the current replica family you specify. If errors occur, the command continues. After finishing, it reports that not all mastership changes succeeded.

With **–long, chmaster** lists the objects whose mastership is changing.

Note: You cannot change the mastership of working schema repository to a different site with the -all option. If you want to change mastership of schemas and other administrative tasks, use the **-workingmaster** option.

-workingmaster

Transfers mastership of the working schema repository to the site you specify. Only the working schema repository can change the mastership of the itself.

You need to have specificed the MASTR family with the **-family** option to use this option.

[**–f·orce** *old-master-replica*]

Warning: Incorrect use of the **–force** form of the command can lead to irreparable divergence among the replicas in a database family.

With **–force**, **chmaster** transfers mastership of all objects in the replica family specified with *old-replica-master.* Use this form of **chmaster** only when replica *old-replica-master* is no longer available (for example, was deleted accidentally).

Examples

- At the boston site, transfer mastership of user group "development" to the sanfrancisco site:

```
multiutil chmaster -clan telecomm -site boston -family
PRODA -user bostonadmin -password secret sanfrancisco
group:development
```

- At replica boston, transfer mastership of user "jsmith" to the sanfrancisco replica:

```
multiutil chmaster -clan telecomm -site boston -family
PRODA -user bostonadmin -password secret sanfrancisco
user:jsmith
```

- At replica paris, transfer mastership of a report format called "Project_report" to the bangalore replica:

```
multiutil chmaster -clan telecomm -site paris -family
PRODA -user parisadmin -password secret bangalore
workspace:\\Public Folders\Triage\project_report
```

- At replica paris, transfer mastership of all items mastered by paris to replica boston:

```
multiutil chmaster -clan telecomm -site paris -family
PRODA -user parisadmin -password secret boston -all
```

- At replica paris which is the current working schema repository site, transfer mastership of all schemas and working schema repository tasks to boston:

```
multiutil chmaster -clan telecomm -site paris -family
MASTR -user parisadmin -password secret boston -working
master
```

```
The working master has been changed from paris to
boston.
```

- In this example, the replica from which you want to transfer mastership is no longer available (perhaps it was accidently deleted and cannot be recovered).

  At replica boston, transfer mastership of all items of replica paris to the boston site. Assume that replica paris is no longer available and use the -force option:

```
multiutil chmaster -clan telecomm -site boston -family
PRODA -user bostonadmin -password secret boston -all
-force paris
```

  Then transfer mastership of all users and groups from the paris replica to the boston replica. Note that you need to indicate the MASTR family:

```
multiutil chmaster -clan telecomm -site boston -family
MASTR -user bostonadmin -password secret boston -all
-force paris
```

  Finally, transfer mastership of the working schema repository from paris to boston:

```
multiutil chmaster -clan telecomm -site boston -family
MASTR -user bostonadmin -password secret boston
-workingmaster -force paris
```

See Also

- describe
- syncreplica

chreplica

Changes the host properties of a replica.

Synopsis

chrep·lica [**–cl·an** *clan-name*] [**–site** *site-name*]
–u·ser *username* [**–p·assword**] *password* [**–host** *hostname replica-selector* |
–size *id-block-size* | **–thresh·hold** *id-block-threshold*]

Description

Use this command when you want to change the shipping server host information or ID block allocation of a replica. Any changes made with the **chreplica** command affect all replicas that reside at that site.

Changing the allocation of ID blocks to a replica

ClearQuest MultiSite controls how many record ID numbers are allocated to each replica. This allocation is done by using ID blocks (groups of IDs).

By default, each replica is given an ID block of 4096 IDs when it is created. When a replica reaches a threshold of 1250 IDs left to use, it is allocated another ID block of 4096 IDs. This ensures that all IDs are unique. ID block allocation is handled internally by the master schema repository site during synchronization operations.

Depending on the activity level of a replica family, it may be helpful to increase the size of the ID blocks that are allocated to a replica, to ensure that synchronization flows smoothly. For example, with the default settings, if a synchronization packet contains enough new records to be added to a replica that the receiving replica exceeds the number of IDs remaining in its current ID block, synchronization will fail.

In order to control how many IDs a replica is allocated, you can use the **-size** option combined with the **-threshhold** options when you create a replica with the **mkreplica -export** command or you can modify these settings with the **chreplica** command.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repositiory form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required..

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–host *hostname*

Enter the name of the new shipping server host.

–size *id-block-size*

ID blocks are specified in increments of 100. You can enter any number from 1 to 1023. For example, to specify an ID block of 30,000, use the number 300 or to specify an ID block of 25,000, use the number 250.

Default: 4096 IDs.

–thres·hold *id-block-threshold*

Allocation threshold is specified as a integer, representing a percentage. You can enter any number from 1 to 63. When the number of remaining record IDs to be used drops below the specified percentage of the current ID block size, an additional block will be allocated.

Default: 6 percent.

replica-selector

Specifies the replica to be changed. Use the site name to specify the replica you want to change.

Default: None.

Examples

Associate replica sanfrancisco with the host pacific1.

```
mulitutil -clan telecomm -site sanfrancisco -family
PRODA -user sfadmin -password secret -host pacific1
sanfrancisco
```

See Also

- chmaster
- syncreplica

control_panel

Sets the e-mail parameters for Rational Shipping Server e-mail notification on Windows.

Synopsis

control_panel –admin *admin-email* **–smtp** *smtp_server_host*
[**–enable_shipping_server_email_notification**]

Description

Use this command to set up e-mail notification when you are using the Rational Shipping Server with ClearQuest MultiSite on the Windows platform. This subcommand is not applicable on UNIX.

If you are using ONLY ClearQuest MultiSite:

Use the control_panel subcommand to specify which e-mail address and server you want to use to receive e-mail notification about Rational Shipping Server operations.

If you are using ClearQuest MultiSite AND ClearCase MultiSite:

By default, ClearQuest MultiSite uses the same e-mail address for Rational Shipping Server notification e-mail as ClearCase MultiSite.

- If you want to use the same e-mail address for Rational Shipping Server operations for ClearQuest MultiSite and ClearCase MultiSite, don't use this command to configure e-mail. E-mail notification should be configured according to your ClearCase documentation. ClearCase MultiSite allows you to configure e-mail notification with the ClearCase Properties dialog and the MultiSite Control Panel options.
- If you want to use a different e-mail address for Rational Shipping Server operations originating from ClearQuest MultSite, you'll need to use this command to indicate the e-mail address you want to use.

Options and arguments

–admin

Enter the e-mail address that you wish to use to receive and send e-mail notification for errors and information sent from the Rational Shipping server. The e-mail address should be entered using the following format:

```
<name>@<domain>.<com>
```

–smtp

Enter the name of the SMTP host that is used in conjunction with the adminitrator's e-mail address you indicated with the **-admin** argument. If you are using ClearCase MultiSite, and are indicating a different e-mail address for use with ClearQuest MultiSite, you must use the same SMTP server as is associated with the e-mail address that is used for ClearCase notification.

–enable_shipping_server_email_notification

Use this argument to enable e-mail notification for ClearQuest MultiSite operations that use the Rational Shipping Server.

Default: If this option is not used, ClearQuest MultiSite uses the ClearCase MultiSite e-mail settings. If ClearCase MultiSite is not present, you must use this option to enable e-mail notification for the Rational Shipping Server.

Examples

- Enables e-mail notification for the Rational Shipping Server for ClearQuest MultiSite (ClearCase MultiSite is not present):

```
multiutil control_panel -admin bostonadmin@telecomm.com
-smtp mailserver0.telecomm.com
-enable_shipping_server_email_notification
```

- Sets up a separate e-mail address to use for ClearQuest MultiSite. In this case, the Rational Shipping Server serves both ClearQuest MultiSite and ClearCase MultiSite:

```
multiutil control_panel -admin bostonadmin@telecomm.com
-smtp mailserver0.telecomm.com
```

describe

Describes the mastership properties of a ClearQuest object

Synopsis

describe [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
-u·ser *username* [**–p·assword**] *password* [**–all | –local |** *entity-selector*...]

Description

This command lists the mastering replica of one or more ClearQuest objects in a replica.

Mastership restricts some of the operations you can perform on an object. For more information about mastership, see *Managing mastership* on page 69.

Options and Arguments

–cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be described.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–all

Lists the mastership of all items in the specified family.

–local

Lists only those items which are mastered by the local site in the specified family.

entity-selector

This is used to indicate the object type of which you want to view the mastership properties. The following ClearQuest objects use mastership.

Change Mastership Object	Syntax	Family used
Record	<record_type>:<record_id>	user database family
Users and groups	user:<login_name> or group:<group_name>	user database family
Workspace item	workspace:<query_name>	user database family
Stateless record which has a <site_name> See *Resolving naming conflicts* on page 95	<record_type>:<record_id>.<sitename>	Stateless record which has a <site_name> See *Resolving naming conflicts* on page 95
Workspace item which has a value in the <site_name> field See *Resolving naming conflicts* on page 95	workspace:<query_name>.<site_name>	Workspace item which has a value in the <site_name> field See *Resolving naming conflicts* on page 95
User or group which has a value in the <site_name field. See *Resolving naming conflicts* on page 95	user:<login_name>.<site_name> or group:<group_name>.<site_name>	User or group which has a value in the <site_name field. See *Resolving naming conflicts* on page 95

Default: None.

Examples

- At replica boston, find out which site is the mastering replica for the user group "development":

```
multiutil describe -clan telecomm -site boston -family
PRODA -user bostonadmin -password secret
group:development
```

- At replica boston, find out which site is the mastering replica for the user "jsmith":

```
multiutil describe -clan telecomm -site boston -family
PRODA -user bostonadmin -password secret user:jsmith
```

- At replica paris, find out which site is the mastering replica of a report format called "Project_report"

```
multiutil chmaster -clan telecomm -site paris -family
PRODA -user parisadmin -password secret
workspace:project_report
```

Note: The easiest way to find out which site masters a record is to look at the value for the "ratl_mastership" field for the record in question.

See Also

- chmaster
- syncreplica

dumpoplog

Lists the contents of the specified replica’s oplog.

Synopsis

dumpoplog [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
-u·ser *username* [**–p·assword**] *password* [**–l·ong** | **–s·hort**]
[**–at** *replica*]
[*oplog-ent-order-num...* | [**–from** *oplog-ent-order-num*]
[**–to** *oplog-ent-order-num*]
]
[**–since** *date-time*] [**–reverse**]

Description

Use **dumpoplog** to find out more about a replica’s oplog. The oplog is used to track all database transactions including record changes and schema modifications. Each oplog entry is numbered. You can list (dump) an entire oplog or view just a range of entries. You can specify oplog entries by integer or by date-time or view only entries that originated from a specific replica site.

Options

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the respective replica.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–l·ong | **–s·hort**

Use **-long** to view all columns of the oplog, including information about the schema revision that applies to the packlet data.

Use **-short** to view each database operation that took place.

Default: If no format is specified, -short is used.

–at *replica*

Lists the oplog entries that originated from the replica(s) you specify. Use the site name to specify the replica.

oplog-ent-order-num...

Allows you to enumerate one or more oplog numbers. Oplog numbers are entered as integers.

SPECIFYING THE OPLOG ENTRIES TO LIST: You can list oplog entries according to number or use a date-time format with the **-since** argument.

–from *oplog-ent-order-num...*

Used in conjunction with the **–to-oplog-ent-order-num...** to indicate a range of oplog numbers. Oplog numbers are entered as integers.

SPECIFYING THE OPLOG ENTRIES TO LIST: You can list oplog entries according to number or use a date-time format with the **-since** argument.

–to *oplog-ent-order-num...*

Used in conjunction with the **–from oplog-ent-order-num...** to indicate a range of oplog numbers. Oplog numbers are entered as integers.

SPECIFYING THE OPLOG ENTRIES TO LIST: You can list oplog entries according to number or use a date-time format with the **-since** argument.

–since *date-time*

All oplog entries after the date-time you specify will be listed.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time*

where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** |**yesterday** |**Sunday** | ... |**Saturday** |**Sun** | ... |**Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–reverse

Allows you reverse the order of the list of the oplog entries.

Examples

- Dumps oplog of the boston operations at the *boston* replica. In this example, no date range was given so all oplog entries associated with the *boston* replica are listed:

  ```
  multiutil dumpoplog -clan telecomm -site boston
  -family PRODA -user bostonadmin -password secret
  -short -at boston
  ```

  ```
  Item 1, 05/21/2001 19:24:42: TABLE_PACKLET (epoch BOSTON.1
  initiated 05/21/2001 19:24:42)
  Item 2, 05/21/2001 19:24:42: SPECIAL_PACKLET (epoch
  BOSTON.2 initiated 05/21/2001 19:24:42)
  Item 3, 05/21/2001 19:24:42: SPECIAL_PACKLET (epoch
  BOSTON.3 initiated 05/21/2001 19:24:42)
  Item 4, 05/21/2001 19:28:02: TABLE_PACKLET (epoch BOSTON.4
  initiated 05/21/2001 19:28:02)
  Item 5, 05/21/2001 19:28:02: SPECIAL_PACKLET (epoch
  BOSTON.5 initiated 05/21/2001 19:28:02)
  Item 6, 05/21/2001 19:28:02: SPECIAL_PACKLET (epoch
  BOSTON.6 initiated 05/21/2001 19:28:02)
  Item 7, 05/21/2001 19:28:02: SPECIAL_PACKLET (epoch
  BOSTON.7 initiated 05/21/2001 19:28:02)
  Item 8, 05/21/2001 19:57:04: ACTION_PACKLET (epoch
  BOSTON.8 initiated 05/21/2001 19:57:03)
  Item 9, 05/21/2001 19:57:09: ACTION_PACKLET (epoch
  BOSTON.9 initiated 05/21/2001 19:57:09)
  Item 10, 05/21/2001 19:57:18: ACTION_PACKLET (epoch
  BOSTON.10 initiated 05/21/2001 19:57:18)
  Item 11, 05/21/2001 19:57:23: ACTION_PACKLET (epoch
  BOSTON.11 initiated 05/21/2001 19:57:23)
  Item 12, 05/21/2001 19:57:28: ACTION_PACKLET (epoch
  BOSTON.12 initiated 05/21/2001 19:57:28)
  Item 13, 05/21/2001 20:02:43: TABLE_PACKLET (epoch
  BOSTON.13 initiated 05/21/2001 20:02:43)
  ```

- Dumps the oplog of operations associated with the *bangalore* replica at the *boston* replica. No date range is given so all oplog entries associated with the bangalore replica are shown. Also, in this example, the only oplog entries associated with the bangalore replica are export operations.

  ```
  multiutil dumpoplog -clan telecomm -site boston
  -family PRODA -user bostonadmin -password secret
  -short -at bangalore
  ```

  ```
  Item 13, 05/21/2001 20:00:15: EXPORT_PACKLET to BANGLALORE
  Item 17, 05/21/2001 21:00:13: EXPORT_PACKLET to BANGLALORE
  ```

See Also

- lsepoch
- scruboplog

lsepoch

Lists the current epoch estimates for specified replicas.

Synopsis

lsepoch [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
-u·ser *username* [**–p·assword**] *password* [*replica...*]

Description

If a clan, site or family is not specified, the epoch estimates for all local instances of clans, sites, and families, respectively, are listed. The login must be valid for all instances. The replica’s own epoch row in its matrix represents its actual level. The other rows represent the replica’s best estimate of the other replicas’ levels.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–family *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

replica...

Specifies the replica for which you want to list epoch information. Use the site name(s) to specify the replica(s) you want to list.

Default: Lists the epoch estimate for each replica in the family.

Examples

- Lists epoch estimate in the current replica's (*boston*) for itself and the *bangalore* replica. In this example, you can see that the bangalore replica needs an update from the boston replica.

  ```
  multiutil lsepoch -clan telecomm -site boston -family
  PRODA -user bostonadmin -password secret boston
  bangalore

  Multiutil: Estimates of the epochs from each site
  replayed at site 'bangalore' (@host1):

  bangalore: 950
  boston: 754
  paris: 451

  Multiutil: Estimates of the epochs from each site
  replayed at site 'boston' (@host2):

  bangalore: 950
  boston: 925
  paris: 504
  ```

- Lists epoch estimate in the current replica's (*boston*) for the entire replica family. In this example, the only replicas in the family are *bangalore, boston,* and *paris*.

  ```
  multiutil lsepoch -clan telecomm -site boston -family
  PRODA -user bostonadmin -password secret

  Multiutil: Estimates of the epochs from each site
  replayed at site 'bangalore' (@host1):

  bangalore: 950
  boston: 754
  paris: 451

  Multiutil: Estimates of the epochs from each site
  replayed at site 'boston' (@host2):
  ```

```
bangalore: 950
boston: 925
paris: 504

Multiutil: Estimates of the epochs from each site
replayed at site 'paris' (@host2):

bangalore: 725
boston: 800
paris: 504
```

See also

- chepoch
- recoverpacket
- restorereplica

lspacket

Describes contents of a packet.

Synopsis

lspacket [**–l·ong** | **–s·hort**] [*pname*...]

Description

This command lists a summary of the contents of one or more disk files that contain replica-creation or update packets. By default, the **lspacket** output includes this information:

- Pathname of each packet
- Type of each packet (`Replica Creation` or `Update`)
- Date of generation
- The originating replica
- Clan and database family to which the packet applies
- Replicas for which the packet is intended.
- Packet sequence number (for a disk file storing one part of a logical packet that has been split into multiple physical packets)

Options and Arguments

–l·ong |

In addition to the default information, lists the following information:

- Name of the replica where the packet was created
- Oplog IDs (epoch numbers) that indicate the contents of the packet

–s·hort

Lists only the pathname of a packet.

pname...

One or more pathnames of files and / or directories (for example, pathnames of MultiSite storage bay directories).

Each file you specify is listed if it contains a physical packet (which may be one of several that make up a logical packet). For each directory you specify, **lspacket** lists packets stored in that directory.

Default: Lists all packets in all MultiSite storage bays on the current host.

Examples

- List a single update packet.

  ```
  multiutil lspacket -long
  c:\ms_ship\incoming\osakaupdate2
  ```

  ```
  Multiutil: Checking shipping bay
  'c:\temp\ms_ship\incoming\osakaupdate2

  Multiutil: Packet
  'c:\ms_ship\incoming\osakaupdate2'...

  Multiutil:      Type: 'UPDATE_PACKET'

  Multiutil:      Sent: 2001-05-21 19:28:03

  Multiutil:      From: BOSTON
  (15F8BF94-446E-11D5-AF98-00B0D0682333)

  Multiutil:      Clan: 'TELECOMM'

  Multiutil:      Recipients: OSAKA

  Multiutil:      Family: 'PRODA'
  ```

- List all packets in all of the local host's storage bays.

  ```
  multiutil lspacket -short
  ```

  ```
  Multiutil: Checking shipping bay
  'd:\temp\ms_ship\incoming\*'...

  Multiutil: Packet
  'd:\temp\ms_ship\incoming\mk_LEX_HUB_21-May-01_19-28-0
  1.xml'...

  Multiutil: Packet
  'd:\temp\ms_ship\incoming\sync_SANFRAN2_HUB_21-May-01_
  22-04-09.xml'...
  ```

See Also

- mkreplica
- MultiSite Control Panel
- syncreplica, shipping.conf (UNIX only)

lsreplica

Lists database replicas.

Synopsis

lsrep·lica [**–cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
-u·ser *username* [**–p·assword**] *password* [**–l·ong** | **–s·hort** | **–fmt** *format*
| **–working·master**]
[**–sib·lings** | [**–sib·lings**] **–infa·mily** *family* | *replica*...]

Description

lsreplica lists information on all replicas recorded in the replica database of the current replica (except for deleted replicas). You can either list all replicas in a known clan or known family within a clan. Other replicas may exist, but the packets containing their creation information have not yet been imported at the current replica.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the respective replica.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR.

You can use the **-siblings** argument or the **-siblings -infamily** arguments to list the replicas in a specific family within the clan, as known to you replicated schema repository.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–l·ong

Includes each replica's creation 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.)

LISTING FORMAT. *Default:* Includes creation event information for each replica.

–s·hort

Lists only replica names.

–fmt *format-string*

With -fmt you can customize simple reports on replica information retrieved by lsreplica. Note that **–fmt** is a mutually exclusive alternative to the **–short** and **–long** options.

-fmt uses conversion specifications which identify particular data items to display and specifies its display format. The conversion specification format closely resembles that of the C-language function **printf():**

- Percent sign (%)
- A key letter (lowercase), which indicates the kind of data to display

Unlike **printf()** specifiers, conversion specifications are not replaced by arguments supplied elsewhere on the command line; they are replaced automatically by **multiutil** with field values extracted from replica.

format-string is a character string, composed of alphanumeric characters, conversion specifications, and escape sequences. It must be enclosed in double quotes (").

Conversion specifications:

%h	Host name
%n	Name of replica
%c	Clan name
%f	Family name
%d	Description of replica, if any.
%s	Status of replica
%%	% character
%z	ID block size
%t	ID block threshold

Escape sequences:

\n	**<NL>**
\t	**<TAB>**
\’	Single quote
\\	Literal (uninterpreted) backslash
\nnn	Character specified by octal code

Example

- Mimic the output from **lsreplica –long**. Note the use of single quotes to enclose the format string, which includes literal double quotes.

```
multiutil lsreplica -clan telecomm -site boston
-family PRODA -user bostonadmin -password secret
-fmt "%n %c %f %h %s %d" boston
```

```
Name:boston; Clan:TELECOMM; Family:PRODA;
Host:mw_tsharp; Status:NORMAL; Description:
Production database
```

–working·master

Lists the working schema repository for the site and family you specified.

–sib·lings

Lists the family members of the current replica, but does not list the current replica itself. This option is useful when you are writing scripts that process only sibling replicas.

–infa·mily *family*

Lists the all replicas in the same family of the specified replica. Use the site name to specify the replica. This option can only be used if you specified MASTR with the **-family** option.

Default: None.

replica...

Specifies the replica for which you want to list information. Use the site name(s) to specify the replica(s) you want to list. You can only list replicas who are members of the same family.

Examples

- List the names of all the replicas participating in the family (PRODA) of the current replica (bangalore):

```
lsreplica -clan telecomm -site bangalore -family PRODA
-user badmin -password secret -short
```

```
BANGALORE
BOSTON
PARIS
SANFRAN
```

- Display a -long listing of replicas participating in the family (PRODA) of the current replica (bangalore):

```
lsreplica -clan telecomm -site bangalore -family PRODA
-user badmin -password secret -long
```

```
Name: BANGALORE; Clan: TELECOMM; Family: PRODA; Host:
dbserver; Status: NORMAL; Description:
Name: BOSTON; Clan: TELECOMM; Family: PRODA; Host:
patriot; Status: NORMAL; Description:
Name: PARIS; Clan: TELECOMM; Family: PRODA; Host:
baguette; Status: REMOVED; Description: deactivated by
rmreplica
Name: SANFRAN; Clan: TELECOMM; Family: PRODA; Host:
cablecar; Status: NORMAL; Description:
```

- List the names of all the replicas participating in the replicated schema repository family (MASTR) that participate in the PRODB family:

  ```
  lsreplica -clan telecomm -site bangalore -family MASTR
  -user badmin -password secret -long -infamily PRODB
  ```

  ```
  Name: BANGALORE; Clan: TELECOMM; Family: PRODB; Host:
  dbserver; Status: NORMAL; Description:
  Name: SYDNEY; Clan: TELECOMM; Family: PRODA; Host:
  harbour1; Status: NORMAL; Description:
  ```

See Also

- **mkreplica**

mkorder

Creates a shipping order for use by the store-and-forward facility.

Synopsis

mkorder –dat·a *packet-pname* [**–sc·lass** *storage-class-name*]
[**–pex·pire** *date-time*] [**–not·ify** *e-mail-address*]
[**–shi·p –cop·y** | **–fsh·ip** [**–cop·y**] | **–out** *order-pname*]
destination...

Description

This command creates a shipping order file for an existing packet or any other file. The shipping order is used by the **shipping_server** to send the packet to one or more destinations.

mkorder submits to the store-and-forward facility a packet that was created with **mkreplica –out** or **syncreplica –out**. You can also use **mkorder** to resubmit store-and-forward packets whose shipping orders have expired, and to transmit arbitrary files among sites.

A shipping order must be located in the same directory as its associated packet or file.

Note: The store-and-forward facility deletes a packet after delivering it successfully (except when the destination is the local host). If you use this command to process a file that must be preserved at your site even after delivery to another site, you must specify the **–copy** option.

Default Handling for Packet-delivery Failures.

Default: If a packet cannot be delivered, it is sent through the store-and-forward facility back to the administrator at the site of the originating replica. A mail message is sent to the store-and-forward administrator. This occurs after repeated attempts to deliver the packet have failed, and the allotted time has expired; it can also occur when the destination host is unknown or a data file does not exist. The store-and-forward configuration settings specify the expiration period and the e-mail address of the administrator.

Options and Arguments

–dat·a *packet-pname*

The pathname of the file containing the packet.

Default: None.

SPECIFYING THE OPLOG ENTRIES TO LIST: If *packet-pname* contains a colon character (**:**), **mkorder** changes the colon to a period character (**.**) during processing. This allows packets to be delivered to Windows machines, which do not allow colons within file names.

–sc·lass *class-name*

Specifies the storage class of the packet and shipping order. If you also use **–ship** or **–fship**, **mkorder** looks up the storage class in the store-and-forward configuration settings in the **MultiSite Control Panel** to determine the location of the storage bay to use.

Default: Creates a shipping order in the directory where the *packet-pname* file is located.

If you omit this option but use **–ship** or **–fship**, **mkorder** places the shipping order in the storage bay location specified for the storage class named *default*.

–shi·p

–cop·y

–fsh·ip [**–cop·y**]

Creates a shipping order for the *packet-pname* file. Using **–fship** (force ship) invokes **shipping_server** to send the packet. Using **–ship** places the shipping order in a storage bay.

–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.

–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.

Windows: This option overrides the storage class's Packet Expiration specification in the MultiSite Control Panel. See the **MultiSite Control Panel** reference page for a discussion of this specification, and of delivery retries in general.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**
where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** |**yesterday** |**Sunday** | ... |**Saturday** |**Sun** | ... |**Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*] *yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–not·ify *e-mail-address*

The delivery-failure message is sent to the specified e-mail address.

If a failure occurs on a Windows host that does not have e-mail notification enabled, a message appears in the Windows Event Viewer. The message includes the *e-mail-address* value specified with this option and a note requesting that this user be informed of the status of the operation. For information on enabling e-mail notification, see the **MultiSite Control Panel** reference page.

destination...

One or more host names (which must be usable by hosts in different domains) or IP addresses. When sending a MultiSite packet, you must specify the host where the replica actually resides or is to be created.

Default: None.

Examples

- Create a shipping order for file **p1**, which is located in the cq_default storage bay. Store the shipping order in the same storage bay as **p1**, and specify that the file is to be sent to host **goldengate**. The lines are broken for readability. You must enter the command on a single physical line.

```
mkorder -data -sclass cq_default "c:\temp\outgoing\p1"
-out "c:\temp\outgoing\p1_order" goldengate
```

```
Shipping order "c:\temp\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.

```
mkorder -data -sclass cq_default "c:\temp\outgoing\p1"
-out "c:\temp\outgoing\p1_order" goldengate
```

```
Shipping order "c:\temp\outgoing\p1_order" 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 p1 -sclass cq_default -pexpire 18-May
-fship bangalore
```

See Also

- **mkreplica**
- (Windows) **MultiSite Control Panel**, (UNIX) **shipping.conf**
- **shipping_server**
- **syncreplica**

mkreplica

Creates a database replica.

Synopsis

• Duplicate an existing database, generating a replica-creation packet:

mkrep·lica –exp·ort [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
-u·ser *username* [**–p·assword**] *password* [**-max·size** *size*]
[**-size** *id-block-size*] [**-thresh·hold** *id-block-threshold*]
{
{-sh·ip|-fsh·ip} -wor·kdir *workpath* [**–sc·lass** *storage-class*]
[**–pex·pire** *date-time*] [**–not·ify** *e-mail-addr*] | **–out** *packet-file-pname*
} *hostname*:*site-name ...*

• Use a replica-creation packet to create a new replica, in addition to a new replicated schema repository:

mkrep·lica –imp·ort
{ **–site** *site-name* **–repo·sitory** *db-info* [**–vendor** *<vendor-type>*]
<db-params>
}
{ [**–data·base** *db-info* [**–vendor** *<vendor-type>*] *<db-params>*
[**–c·omments** *comments*] **{** *packet-file path* **|** *packet-dir-path* **}...**

• Use a replica-creation packet to create a new replica within the same clan as the existing replicated schema repository at the current site:

mkrep·lica –imp·ort { [**–cl·an** *clan-name*] [**-site** *site-name*]
–u·ser *username* [**–p·assword**] *password*
{ **–data·base** *db-info* [**–vendor** *<vendor-type>*] *<db-params>*
[**–c·omments** *comments*] **{** *packet-file path* **|** *packet-dir-path* **}...**

where *<db-params> :=*

if <vendor> == DB2
<db-info> := Database Alias
<db-params> := **-dbologin** *dbo-name* [*dbo-pwd*]

if <vendor> == ORACLE
*<db-info> := SQL*Net Alias*
<db-params> := **-dbologin** *dbo-name dbo-pwd* [**-connectopts**
connect-options]

if <vendor> == SQL_SERVER
<db-info> := Physical Database Name
<db-params> := **-server** *server-name* **-dbologin** *dbo-name* [*dbo-pwd*]
-rwlogin *rw-login* [*rw-pwd*] [**-rologin** *ro-login* [*ro-pwd*]]

Description

The creation of a new database replica is a two-phase process. Both phases require you to enter a **mkreplica** command.

Note: Before replicating a the first database of a clan, you must first **activate** the database set to which it belongs.

This command may take a long time. The database is locked while an export is in progress. For more information see *Exporting a replica* on page 39.

1. The **mkreplica –export** command duplicates the contents of the specified user database and its associated schema repository. This generates a single logical *replica-creation packet* for transmission to one or more other sites. As described in *Replica-creation packets* on page 164, 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 file.)

   SPECIFYING THE OPLOG ENTRIES TO LIST: Creating multiple replicas in one **mkreplica –export** command is more efficient than using multiple **mkreplica –export** commands.

2. At each new site, a **mkreplica –import** command uses the replica-creation packet to create a new replica. The new replica consists of two replicated databases, a schema repository and a user database. This command varies if you are adding a user database replica which belongs to a family within the same clan of an existing replicated schema repository at your site.

Creating empty vendor databases

At each new site, the site administrator needs to have created empty vendor databases to receive the replica data. If this is the first replica at a site, you need at least two empty vendor databases, one for the replicated schema repository and one for the user database replica.

Note: If you are adding a user database replica family that is in the same clan as an already existing replicated schema repository at your site, you don't need to create a vendor database for the replicated schema repository. You can associate the new user database replica with the existing replicated schema repository at your site.

Oplog information

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

Note: Before entering a **mkreplica –export** command, make sure ClearQuest MultiSite licenses are installed at the original site. After you enable replication of the original database, developers cannot access the database without a ClearQuest MultiSite license (in addition to a ClearQuest license). A ClearQuest MultiSite license is also required to run **mkreplica –export**.

Allocating ID blocks to a replica

ClearQuest MultiSite controls how many record ID numbers are allocated to each replica. This allocation is done by using ID blocks (groups of IDs).

By default, each replica is given an ID block of 4096 IDs when it is created. When a replica reaches a threshold of 1250 IDs left to use, it is allocated another ID block of 4096 IDs. This ensures that all IDs are unique. ID block allocation is handled internally by the master schema repository site during synchronization operations.

Depending on the activity level of a replica family, it may be helpful to increase the size of the ID blocks that are allocated to a replica, to ensure that synchronization flows smoothly. For example, with the default settings, if a synchronization packet contains enough new records to be added to a replica that the receiving replica exceeds the number of IDs remaining in its current ID block, synchronization will fail.

In order to control how many IDs a replica is allocated, you can use the **-size** option combined with the **-threshhold** options when you create a replica with the **mkreplica -export** command or you can modify these settings with the **chreplica** command.

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.) The packet carries one or more replica specifications, each of which indicates the host on which a new replica is to be created, along with the new replica's name.

The user database and schema repository are locked during the -export phase.

The **–maxsize** option divides the single logical packet into multiple physical packets (disk files or tapes) to conform with limitations of the transfer medium.

Recovering from failed imports

If a replica import is interrupted or fails for any reason (power outage, etc.), you'll need to delete the vendor database(s) and create new vendor database for the failed import operation and re-run mkreplica -import.

It is possible to have a successful import of the replicated schema repository, but a failed import of the user database replica. In this case, you only need to delete re-create the vendor database that was intended for the user database replica. For more information, see *Recovering from a failed replica import* on page 42.

Cleaning up used packets

Replica-creation packets are not deleted after import. The ClearQuest MultiSite administrator at the new replica site must delete replica-creation packets after importing them with **mkreplica –import**.

Error Handling for Packet Delivery Failures

If a packet cannot be delivered, it is sent through the store-and-forward facility back to the administrator at the site of the originating replica. A mail message is sent to the store-and-forward administrator. This occurs after repeated attempts to deliver the packet have all failed, and the allotted time has expired; it can also occur when the destination host is unknown or a data file does not exist. The store-and-forward configuration settings specify the expiration period and the e-mail address of the administrator.

Restrictions

Locks: This command fails if the database is locked (i.e., during the upgrade process) or while another ClearQuest MultiSite operation is being performed.

Other restrictions:

- You cannot replicate a database to a host running a different version of ClearQuest MultiSite.

Options and Arguments

Export Phase

The following sections describe the options and arguments for use with **mkreplica –export**. The following arguments specify the database to be replicated.

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be created. The site name is specified when the database is replicated.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: Not applicable. When you run mkreplica the associated schema repository of the user database family you specify is automatically included in the replica-creation packet.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. You must have Super User privileges to administer ClearQuest MultiSite.

–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

Specifying the replica-creation packet size: Default: None. 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 **MultiSite Control Panel**.
- Packets created with **–out** are no larger than 2 GB.

The **mkreplica** command fails if it creates a packet larger than the size supported by your system.

–c·omments *comments*

Add any comments you want to store with this replica's information.

–size *id-block-size*

ID blocks are specified in increments of 100. You can enter any number from 1 to 1023. For example, to specify an ID block of 30,000, use the number 300 or to specify an ID block of 25,000, use the number 250.

Default: 4096 IDs.

–thres·hold *id-block-threshold*

Allocation threshold is specified as a integer, representing a percentage. You can enter any number from 1 to 63. When the number of remaining record IDs to be used drops below the specified percentage of the current ID block size, an additional block will be allocated.

Default: 6 percent.

–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**.

Default disposition of replica-creation packet: None. You must specify how the replica-creation packet created by **mkreplica –export** is to be stored and / or transmitted to other sites.

SPECIFYING THE OPLOG ENTRIES TO LIST: 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 replica-creation packet.

–wor·kdir *workpath*

A directory for use by **mkreplica** as a temporary workspace; it is deleted when **mkreplica** finishes. This directory must not already exist.

This directory is needed only when using **-ship** or **-fship**.

Default Location for temporary Workspace: None.

–sc·lass *storage-class*

Specifies the *storage class* of the packet and shipping order. **mkreplica** looks up the storage class in the store-and-forward configuration settings in the **MultiSite Control Panel** to determine the location of the storage bay to use.

Default: If you omit this option, **mkreplica** places the packet in the storage bay location specified for the *cq_default* class.

–out *packet-file-pname*

Places the first physical replica-creation packet in file *packet-file-pname*. Additional packets are placed in files named *packet-file-pname_2*, *packet-file-pname_3*, and so on.

The replica-creation packets are not delivered automatically; use an appropriate mechanism (for example, electronic mail, ftp, or postal service) to deliver them.

You can create a packet using **–out**, and subsequently deliver it using the store-and-forward facility. See the **mkorder** reference page for details.

–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 storage class's Packet Expiration specification in the MultiSite Control Panel. See the **MultiSite Control Panel** reference page for a discussion of this specification and of delivery retries in general.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time* | **now**

where:

date	:=	*day-of-week* **|** *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** |**yesterday** |**Sunday** | ... |**Saturday** |**Sun** | ... |**Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–not·ify *e-mail-address*

The delivery-failure message is sent to the specified e-mail address.

If a failure occurs on a Windows host that does not have e-mail notification enabled, a message appears in the Windows Event Viewer. The message includes the *e-mail-address* value specified with this option and a note requesting that this user be informed of the status of the operation. For information on enabling e-mail notification, see the **MultiSite Control Panel** reference page.

hostname:site-name...

One or more arguments, each of which indicates one new replica to be created from this packet at another site.

Default: None.

hostname	Names the machine where the new replica's storage directory will be created. *hostname* must be usable by hosts in different domains. It is used by the ClearQuest *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 Windows NT, the computer name is displayed in the **Network Settings** dialog box, which is accessible from the **Network** icon in the Control Panel. On Windows 2000, the computer name is displayed on the **Network Identification** tab in the **System Properties** dialog box, which is accessible from the **System** icon in the Control Panel.
site-name	Name by which the replica will be identified in multiutil commands. The site name must be an identifier and can be up to 50 characters long. This name must be unique within the respective clan. In other words, there cannot be two sites with the same name participating in the same clan.

Options and Arguments

Import Phase

The following sections describe the options and arguments for use with **mkreplica –import**.

Before using **mkreplica -import**, the site administrator needs to have created empty vendor databases to receive the replica data.

Importing both a replicated schema repository and a user database replica

If there is no existing replicated schema repository at the site to which you are importing the replica, you need to enter the repository name, site name and vendor database parameters.

–site *site-name*

The name of the site where the replica will be imported. The site name was given to the replica when it was exported. If you don't know the site name, contact your ClearQuest MultiSite administrator. This name must match the site name given to the replica when it was exported.

–repo·sitory *db-info*

Enter the database information respective to the vendor database you are using.

Vendor Database	<dbinfo> value
DB2	*Database Alias*
Oracle	*SQL*Net Alias*
SQL Server	*Physical Database Name*

–vendor *<vendor-type>*

Enter the database vendor you are using. Supported vendor types are: DB2, SQL_SERVER, and ORACLE.

<db-params>

The required database parameters are the same parameters needed to connect to any ClearQuest database. Note these when you create the vendor database to which you import the replica. For more information about how to create empty vendor databases and the parameters you will need, see the *Installing Rational ClearQuest* documentation.

SPECIFYING THE OPLOG ENTRIES TO LIST: When you import a replica, you'll need to specify the database parameters of both the vendor database that will hold the replicated schema repository and the vendor database that will hold the user database replica. You must have first created these databases before importing a replica packet.

here *<db-params> :=*

if <vendor> == DB2
<db-info> := Database Alias
<db-params> := **-dbologin** *dbo-name* [*dbo-pwd*]

if <vendor> == ORACLE
*<db-info> := SQL*Net Alias*
<db-params> := **-dbologin** *dbo-name dbo-pwd* [**-connectopts** *connect-options*]

if <vendor> == SQL_SERVER
<db-info> := Physical Database Name
<db-params> := **-server** *server-name* **-dbologin** *dbo-name* [*dbo-pwd*] **-rwlogin** *rw-login* [*rw-pwd*] [**-rologin** *ro-login* [*ro-pwd*]]

Note: The read-only login (-rologin) for SQL_SERVER is needed only when creating a replicated schema repository.

–data·base *db-info*

Enter the user database information respective to the vendor database you are using.

Vendor Database	**<dbinfo> value**
DB2	*Database Alias*
Oracle	*SQL*Net Alias*
SQL Server	*Physical Database Name*

–vendor *<vendor-type>*

Enter the database vendor you are using. Supported vendor types are: DB2, SQL_SERVER, and ORACLE.

Default: Uses the same vendor as specified for the **-repository**.

<db-params>

The required database parameters are the same parameters needed to connect to any ClearQuest database. Note these when you create the vendor database to which you import the replica. For more information about how to create empty vendor databases and the parameters you will need, see the *Installing Rational ClearQuest* documentation.

SPECIFYING THE OPLOG ENTRIES TO LIST: When you import a replica, you'll need to specify the database parameters of both the vendor database that will hold the replicated schema repository and the vendor database that will hold the user database replica. You must have first created these databases before importing a replica packet.

here *<db-params> :=*

if <vendor> == DB2
<db-info> := Database Alias
<db-params> := **-dbologin** *dbo-name* [*dbo-pwd*]

if <vendor> == ORACLE
*<db-info> := SQL*Net Alias*
<db-params> := **-dbologin** *dbo-name dbo-pwd* [**-connectopts** *connect-options*]

if <vendor> == SQL_SERVER
<db-info> := Physical Database Name
<db-params> := **-server** *server-name* **-dbologin** *dbo-name dbo-pwd* **-rwlogin** *rw-login* [*rw-pwd*] [**-rologin** *ro-login* [*ro-pwd*]]

The read-only login (-rologin) for SQL_SERVER is needed only when creating a replicated schema repository.

--c·omments *comments*

Enter any comments you want to include.

packet-file-path | packet-dir-path ...

Specifies a pathname of a replica-creation packet created by **mkreplica –export**. For a logical packet that spans multiple disk files, **mkreplica** scans the directory containing *packet-file-pname* for related physical packets.

If you also specify one or more *packet-dir-path* arguments, **mkreplica** searches for additional packets in these directories.

Default: None.

Adding an additional replica to an existing clan (replicated schema repository)

If you are adding a replica family to an existing clan (replicated schema repository), you only need to create a vendor database for the user database replica. Instead of entering the database parameters for the a new replicated schema repository, you can use the clan name and indicate the following login information.

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be imported. The site name is specified when the database is replicated.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–data·base *db-info*

Enter the user database information respective to the vendor database you are using.

Vendor Database	**<dbinfo> value**
DB2	*Database Alias*
Oracle	*SQL*Net Alias*
SQL Server	*Physical Database Name*

–vendor *<vendor-type>*

Enter the database vendor you are using. Supported vendor types are: DB2, SQL_SERVER, and ORACLE.

<db-params>

The required database parameters are the same parameters needed to connect to any ClearQuest database. Note these when you create the vendor database to which you import the replica. For more information about how to create empty vendor databases and the parameters you will need, see the *Installing Rational ClearQuest* documentation.

SPECIFYING THE OPLOG ENTRIES TO LIST: When you import a replica, you'll need to specify the database parameters of both the vendor database that will hold the replicated schema repository and the vendor database that will hold the user database replica. You must have first created these databases before importing a replica packet.

<db-params> :=

if <vendor> == DB2
<db-info> := Database Alias
<db-params> := **-dbologin** *dbo-name* [*dbo-pwd*]

if <vendor> == ORACLE
*<db-info> := SQL*Net Alias*
<db-params> := **-dbologin** *dbo-name* [*dbo-pwd*] [**-connectopts** *connect-options*]

if <vendor> == SQL_SERVER

<db-info> := Physical Database Name
<db-params> := **-server** *server-name* **-dbologin** *dbo-name dbo-pwd* **-rwlogin** *rw-login* [*rw-pwd*] [**-rologin** *ro-login* [*ro-pwd*]]

The read-only login (-rologin) for SQL_SERVER is needed only when creating a replicated schema repository.

--c·omments *comments*

Enter any comments you want to include.

packet-file-path **|** *packet-dir-path ...*

Specifies a pathname of a replica-creation packet created by **mkreplica –export**. For a logical packet that spans multiple disk files, **mkreplica** scans the directory containing *packet-file-pname* for related physical packets.

If you also specify one or more *packet-dir-path* arguments, **mkreplica** searches for additional packets in these directories.

Default: None.

Examples

Exports

- Generate a replica-creation packet from the *boston* site of the *PRODA* database family, which will be used at remote host *sushi* to create a new replica named *osaka*. Place the packet in a file in directory c:\temp. The lines are broken for readability. You must enter the command on a single line.

```
multiutil mkreplia -export -clan telecomm -site boston
-family PRODA -user bostonadmin -password secret
-c "make a new replica for osaka" -out osakareplica
sushi:osaka
```

- Similar to the proceeding example but place the packet file in a storage bay, for shipping at some later time by the store-and-forward facility. The lines are broken for readability. You must enter the command on a single line.

```
multiutil mkreplia -export -clan telecomm -site boston
-family PRODA -user bostonadmin -password secret
-c "make a new replica for osaka" -ship -workdir
c:\temp\working -sclass cq_default -pexpire
22-November-2003 -nofify admin@company.com
```

Imports

- Imports a new database replica osaka and its associated replicated schema repository into SQL Server databases.

  ```
  multiutil mkreplica -import -osaka -repository
  cq_schemarepo -vendor SQL_SERVER -server unagi
  -dbologin japanadmin secret -rwlogin japanadmin secret
  -rologin japanadmin secret
  -database cq_userdb -vendor SQL_SERVER -dbologin
  juseradmin secret -rwlogin juseradmin secret
  ```

- Imports a new user database replica that is part of the telecomm clan at the osaka site. The new user database replica is being imported into a SQL Server database.

  ```
  multiutil mkreplica -import -clan telecomm -site osaka
  -user cqjapanadmin -p secret
  -database cq_userdb -vendor SQL_SERVER -dbologin
  juseradmin secret -rwlogin juseradmin secret
  ```

See Also **activate**

MultiSite Control Panel

Configures store-and-forward facility.

Synopsis

%SystemRoot%\System32\ms.cpl

To open the MultiSite Control Panel, double-click the MultiSite icon in Windows Control Panel.

Description

The **MultiSite Control Panel** controls operation of the MultiSite store-and-forward facility on each host. MultiSite installation creates registry keys in which all these entries are defined. In some cases, the corresponding store-and-forward operation fails if a parameter is not defined, and in other cases there is a hard-coded default.

The **MultiSite Control Panel** provides controls for setting the configuration parameters described in the following sections.

Maximum Packet Size

This setting controls the splitting of individual logical packets into multiple physical packets. This value specifies the maximum size for a physical packet file. Limiting the size of physical packets can improve the reliability of packet delivery in some networks. To specify no limit, use 0 (zero).

This value is used by the following commands (unless you also specify **–maxsize**):

- **mkreplica –fship**
- **mkreplica –ship**
- **syncreplica –fship**
- **syncreplica –ship**

When you invoke **mkreplica** or **syncreplica** with **–out** this value is not used, and you must use **–maxsize** to limit the packet size.

Default: 2097151KB

Administrator E-mail

E-mail notification is used when the following 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 it’s sending host.
- syncreplica -import finds a replica-creation packet.

To enable e-mail notification on Windows, see *control_panel* on page 139.

Default: None.

Storage Classes

Storage Class Name

This setting specifies the name of a storage class.

You can associate values for packet expiration, the storage bay, the return bay, and the receipt handler with each storage class.

Note: You cannot use the same storage class for both ClearQuest MultiSite and ClearCase MultiSite.

Default: The default storage class used by ClearQuest MultiSite varies according to command. All ClearQuest MultiSite commands that use the -sclass argument use the default storage class name of *cq_defaul*t, except **mkorder** and **shipping_server**, which default to the storage class name of *default*.

Packet Expiration

This setting specifies the expiration period (in days) for shipping orders generated in the associated storage class. This period begins at the time the shipping order is generated.

If a packet cannot be delivered to all its specified destinations in the specified number of days, the packet is returned to the original sending host and a message is sent to the address specified in the **Administrator Email** box.

E-mail notification for ClearQuest MultiSite requires the use of the control_panel subcommand, see *control_panel* on page 139.

If e-mail notification is not enabled, a message is written to the Windows Event Viewer.

A value of 0 (zero) specifies no expiration; delivery is reattempted indefinitely.

This setting is overridden by the **–pexpire** option to **syncreplica** or **mkreplica**.

Packet re-delivery

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.

Default: When the **Use Default Expiration** check box is selected, the storage class uses the packet expiration value associated with the *cq_default* class. (This value is not shown in the **Packet Expiration** box; you must display the *cq_default* class to determine the value.)

Storage Bay Path

This setting defines the location of a storage bay, a directory that holds the outgoing and incoming update packets and shipping orders of a particular storage class.

Packets placed in a storage bay on an NTFS file system inherit the Windows ACL on the storage bay. Define ACLs on the storage and return bays to enable successful execution of MultiSite commands to process the packets, and to guard against unauthorized access.

Packets stored on FAT file systems have no protections.

Before using the store-and-forward facility, make sure that the disk partition where you will have your storage bays has sufficient free space for anticipated replica-creation and update packets. The amount of available space on the disk partition where the shipping and return bays are located must be at least twice as big as the largest packet that will be stored in the bays. This is because there may be two copies of the same packet in the bay at one time: one on its way to another destination, and another waiting to be applied to the replica on the host.

Note: When you create a new storage class, the storage bay and return bay that you specify are created, along with the *incoming* and *outgoing* directories within the bays.

Return Bay Path

This setting defines a return bay (directory) to hold incoming or outgoing packets in the process of being returned to their origin because they could not be delivered to all specified destinations.

Packets placed in a return bay inherit the ACL on the directory.

Receipt Handler Path

This setting 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 if you are using a scheduling program to automate the sending and applying of packets. By default, no file is specified.

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 *cq_default* storage class, **shipping_server** uses the batch file or program specified for *cq_default*.
2. Invokes the receipt handler, as follows:

   script-pname [**–d·ata** *packet-file-pname*] [**–a·ctual** *shipping-order-pname*] [**–s·class** *storage-class*] **–o·rigin** *hostname*

where

script-pname	Script specified in the **RECEIPT-HANDLER** entry.
–d·ata*packet-file-pname*	Location of the packet. This parameter is used only when the packet is destined for this host.
–a·ctual *shipping-order-pname*	Location of the shipping order. This parameter is used only when the packet is destined for another host.
–s·class *storage-class*	Storage class associated with the packet. This parameter is used only if the packet was associated with a storage class when it was created.
–o·rigin *hostname*	Host name of the machine from which the packet was first sent.

Note: If a packet is destined for both the local host and another host, both the **–data** and **–actual** parameters are used. The packet is imported at the replica on the host, then forwarded to its next destination.

Default: None.

ROUTING INFORMATION

The Routing Information settings control the network routing of packets.

Next Routing Hop

The host specified here is the next destination for packets whose final destination is any of the host names specified in the **Destination Hostnames** list. This host is responsible for delivery of the packet to its destinations. You can specify a host using either its host name (which must be usable by hosts in different domains) or its numeric IP address.

Default: None.

Destination Host Names

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

Default: None.

recoverpacket

Resets epoch row table so changes in lost packets are resent.

Synopsis

recoverpacket [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name* **-u·ser** *username* [**–p·assword**] *password* [**–sin·ce** *date-time*] *replica* ...

Description

The **recoverpacket** command resets the epoch row at a sending replica to reflect the last synchronization sent to a receiving replica before a particular time. It scans through a list of epoch rows saved at the time of each export, looking for an entry prior to the time specified. When it finds this entry, it uses the associated row to reset the epoch row for just the specified receiving replica. The next time a packet is sent, it includes the changes that were in the lost packet.

Resetting Epoch Numbers Automatically

When you send an update packet to another replica, success of the transport and import phases is assumed. Therefore, the sending replica's epoch number matrix is updated to reflect that the changes are made at the receiving replica. However, if the packet is lost before reaching the receiving replica, the sending replica's assumption that the receiving replica is up to date is incorrect.

The updated epoch numbers must be returned to the values they had before the packet was sent. Making these corrections to the sending replica's epoch number matrix causes it to include the same changes in the next update packet it sends to the receiving replica.

The administrator at the receiving replica needs to run an dumpoplog command to determine the time of the last successful import. The administrator at the sending replica uses this time in the **recoverpacket** command.

Note: If the two sites are not in the same time zone, or you do not send packets at the same time you generate them (for example, you generate packets at midnight and send them at 6:00 A.M.), you must adjust for the time.

Resetting Epoch Numbers Manually

If there are no saved epoch rows for the replica that are as old as the specified time, the **recoverpacket** command fails. In this case, the administrator at the receiving site must use the **lsepoch** command to determine the correct epoch number, and the administrator at the sending site must run **chepoch** on the sending replica to reset the epoch row. For more information, see *chepoch* on page 128.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: Use the MASTR family if you want to recover information about the working schema repository. If you've lost a packet, run **recoverpacket** on both the MASTR and user database families, then re-execute **syncreplica**.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–since *date-time*

All oplog entries after the date-time you specify will be listed.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time*

where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** |**yesterday** |**Sunday** | ... |**Saturday** |**Sun** | ... |**Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

replica...

Specifies the replica for which the epoch row is reset. Use the site name to specify the replica.

Examples

- At the *boston* site, reset the epoch row for replica *sanfran_hub* so that changes sent since February 20, 2001 will be included in the next packet that is sent.

```
multiutil recoverpacket -clan telecomm -site boston
-family PRODA -user admin -p "" -since 02-20-2001
sanfran_hub
```

See Also

- **chepoch**
- **lsepoch**
- **restorereplica**

restorereplica

Recovers packets generated since the restoration of a replica from backup.

Synopsis

restorereplica [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name* **-u·ser** *username* [**–p·assword**] *password* [**-force**] [**–completed**] [**–replace**] [*replica ...*]

Description

Execute this command IMMEDIATELY after restoring a replica from backup. Proceeding with normal development (and generating new changes) at a restored replica before executing this command can lead to IRREPARABLE inconsistencies among the replicas in a replica family.

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

1. Causes the current replica to create special update packets that contain update requests to other replicas. These packets are sent to other replicas via with **syncreplica -export**.
2. Locks the current replica and marks the replica as being in the process of restoration.
3. Requests the recovery information for the replica.
4. Causes **lsreplica –long** to indicate which replicas must send restoration updates to the current replica.

The current replica remains in the restoration state until your site has received and applied (using **syncreplica –import**) all the restoration updates needed to bring the replica up to date with the state of the 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. 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 replicas* on page 97.

Locking Of The Replica

restorereplica locks the current replica This 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 and unlocks the replica.

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.

The **restorereplica** command requires restoration updates from *all* other members of its family. The **syncreplica** command declares the replica to be restored completely only after all the updates have been processed.

Warning: Making a mistake in using this optimization can make the restored replica irreparably inconsistent with other replicas.

Restrictions

Locks: The database is locked during the restore operation. Read-only commands such as lsepoch and lsreplica can still access the database.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: Not applicable. Restoring a member of user database family automatically requests updates and its associated replicated schema repository, if necessary.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–f·orce

Suppresses the confirmation step.

Default: **restorereplica** prompts you for confirmation.

Warning: Incorrect use of this option allows new changes to be made to the replica before all missing changes are received from other replicas. This may place the entire replica family in an irreparably inconsistent state.

–completed

Use this option when you are finished restoring the replica. This option marks the replica as restored and unlocks the database. If this option is used, no more packet requests can be sent and no more restoration packets can be played at this replica.

–rep·lace *replica*...

Specifies the subset of replicas from which restoration updates are required. Use the site name(s) to specify the replica(s) from which you want to receive restoration packets.

Examples For an example of restoring a replica, see *Restoring replicas* on page 97.

See Also

- **chepoch**
- **lsepoch**
- **lsreplica**
- **syncreplica**

rmreplica

Deletes a replica.

Synopsis

rmreplica [**–cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name* **-u·ser** *username* [**–p·assword**] *password* [**–dbset** *new-name*] *replica*

Description

Warning: To delete a replica, you must complete all steps described in *Removing a replica* on page 64. If you do not complete all steps in the correct order, synchronization and mastership problems can occur in other replicas in the database family.

This command deletes from the current replica's database the database-replica record that records the existence and identity of another replica. Typically, you use this command at your site to record the fact that a replica at another site has been decommissioned and deleted.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: Not applicable. If there is only one user database family at the site specified, this command removes the replicated schema repository as well. If there is more than one user database family at the site, the schema repository will not be removed.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–dbset *new-name*

This option is only used when removing the last replica of a clan. When you remove the last replica of a clan, you need to rename the dbset name so it does not include any ClearQuest MultiSite flags.

replica

Specifies the replica to be deleted from the current replica's database. Use the site name to specify the replica.

Default: Defaults to the current replica when running the command from the location of the replica to be removed. This argument is mandatory when running the command from any other replica location.

Examples

- Remove the replica record that records the existence of replica **paris** from the database of the current replica site, Boston.

```
multiutil rmreplica -clan telecomm -site Boston -family
PRODA -user admin -password "" paris
```

- Remove the last replica of a clan and return the existing replica to a non-replicated state.

```
multiutil rmreplica -user admin -password "" -dbset
cqdatabase
```

See Also

- **chmaster**
- **mkreplica**

scruboplog

Deletes oplog entries associated with the respective replica.

Synopsis

scruboplog [–cl·an *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
[**-u·ser** *username*] [**–p·assword**] *password* **–before** { *date-time* |
oplog-number }

Description

Oplog entries must be kept in the replica for a significant period. In the near term, they are required when the replica generates update packets to be sent to all other replicas. Beyond that, oplog entries may be required to help other replicas recover from failures.

However, you may want to delete (scrub) oplog entries occasionally to optimize hard drive space where the replica resides. You can also use the scruboplog command to delete the oplog of a replica which will no longer be used.

Although oplog entries only record the changes that have taken place in your database, overtime this information could require as much space as the data itself.

Before scrubbing (deleting) oplog entries for a replica, you should be sure that they are no longer needed and that each replica has the current information from the replica’s oplog that you want to delete.

Options and Arguments

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.

- **replicated schema repository family**: To specify the family of the replicated schema repository use the family name, MASTR.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–before { *date-time* | *oplog-number* }

You can either enter the date-time of the oplog you want to delete or you can specify a number. All oplog entries previous to the date-time or oplog number you specify will be deleted.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time*

where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** |**yesterday** |**Sunday** | ... |**Saturday** |**Sun** | ... |**Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
```

```
8-jun
13:00
today
9-Aug.10:00UTC
```

The *oplog-number* argument should be entered as an integer. If you're unsure of what oplog number to specify, use dumpoplog to find out more information about the replica's oplog you want to delete.

Examples

- Deletes the oplog for the PRODA family of the boston replica, from January 24, 2001 and earlier.

  ```
  multiutil dumpoplog -clan telecomm -site boston -family
  PRODA -user bostonadmin -password secret -before
  01/24/2001
  ```

- Deletes the oplog for the PRODA family of the boston replica from epoch number 56 and earlier.

  ```
  multiutil dumpoplog -clan telecomm -site boston -family
  PRODA -user bostonadmin -password secret -before 56
  ```

See Also

- **dumpoplog**
- **syncreplica**

shipping.conf

Store-and-forward configuration file

Synopsis

/var/adm/atria/config/shipping.conf

Description

This file controls the operation of the MultiSite store-and-forward facility on each host. The file consists of comment lines (starting with #) and one or more configuration entries.

The **shipping.conf** file can contain the configuration entries described below. In some cases, the corresponding store-and-forward operation fails if an entry is missing; in other cases, there is a hard-coded default.

MultiSite installation creates the file ccase-home-dir/config/services/shipping.conf.template, in which all these entries are defined. If /var/adm/atria/config/shipping.conf does not exist, the installation creates it by copying the template file. If /var/adm/atria/config/shipping.conf exists, the installation advises you to compare the existing file to the template and make any necessary changes.

NOTE: If you do not install ClearCase and MultiSite in the default installation directory (/usr/atria), you must edit the **shipping.conf** file and change /usr/atria to the pathname of your installation directory.

Packet size

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

Controls the splitting of individual logical packets into multiple physical packets. Limiting the size of physical packets can improve the reliability of packet delivery in some networks. The *size* integer (with the optional **k**, **m**, or **g** suffix) specifies the maximum size for a physical packet file. **k** specifies KB (kilobytes); **m** specifies MB (megabytes); **g** specifies GB (gigabytes). Omitting the keyletter 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**

When you invoke **mkreplica** or **syncreplica** with **–out**, this value is not used and you must use **–maxsize** to limit the packet size.

Default: 2097151k

Notification

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

The electronic mail program to be invoked in these circumstances:

- When **shipping_server** finds that a shipping order it is about to process has expired
- When an undeliverable packet is returned to the original sending host by another host's **shipping_server** (see the description of **EXPIRATION**)
- When **syncreplica –import** finds a replica-creation packet, which must be processed with a **mkreplica** command

The mail program is invoked as follows:

e-mail-program-pathname **–s** *subject* **–f** *message-file addr* **. . .**

Default: The electronic mail program specified in the default configuration file is **/usr/atria/bin/notify**. (This program is also used if no **NOTIFICATION-PROGRAM** entry exists.)

Administrator E-mail

ADMINISTRATOR *e-mail-address*

The electronic mail address of the administrator who administers the store-and-forward facility on the local host.

A mail message is sent to the specified address in the circumstances listed in **Notification**. The configuration file can contain multiple **ADMINISTRATOR** entries; messages are sent to all the specified mail addresses.

Default: If there is no **ADMINISTRATOR** entry, mail is sent to **root**.

STORAGE BAY

STORAGE-BAY *storage-class directory-pathname*

Defines a storage bay, a directory that holds the outgoing and incoming update packets and shipping orders of a particular storage class.

You can use multiple **STORAGE-BAY** entries to define multiple storage bays for a particular storage class. In this case, **shipping_server** selects one of the bays for each incoming packet based on the available disk space in the bays' disk partitions. The order in which you specify storage bays does not matter.

Storage classes for ClearCase MultiSite

MultiSite installation establishes a default storage bay and return bay on the local host in the **/var/adm/atria/shipping** directory. These bays contain subdirectories named **incoming** and **outgoing**, which hold incoming and outgoing packets. Shipping operations look for packets in these subdirectories.

Storage classes for ClearQuest MultiSite

You'll need to add a lines for the storage bay path for ClearQuest MultiSite. You'll also need to specify the name of the storage class that you are using.

For example,

```
# Storage bay path for ClearQuest MultiSite
STORAGE-BAY cq_default /people/jsmith/shipping/ms_ship/
RETURN-BAY cq_default /people/jsmith/shipping/ms_rtn
```

Before using the store-and-forward facility, make sure that the disk partition where the shipping directory is created has sufficient free space for anticipated replica-creation and update packets.

You must create *directory-pathname* with a standard UNIX **mkdir** command. You must also create the **incoming** and **outgoing** directories within the new storage bay. Packets placed in a storage bay by the **shipping_server** are assigned the same owner, groups, and read-write permissions as the storage bay itself. (Execute permission and any special permissions on the storage bay are ignored.) Be sure to adjust these permissions (if necessary) to enable successful execution of MultiSite commands to process the packets, and to guard against unauthorized access.

Note: The **incoming** and **outgoing** directories must be on the same file system.

Default: Specifying **–default** as the storage class specifies a directory to be used for packets that are not assigned to any storage class, and for packets whose storage class is not configured. The default configuration file includes this line:

```
STORAGE-BAY –default /usr/atria/shipping/ms_ship
```

Return bay

RETURN-BAY *storage-class directory-pathname*

Defines a return bay (directory) to hold incoming or outgoing packets in the process of being returned to their origin because they could not be delivered to all specified destinations.

Return bays are like storage bays in several aspects:

- Each storage class can have multiple **RETURN-BAY** entries, and order does not matter.
- You must create each return-bay directory with a standard UNIX **mkdir** command.
- You must create the **incoming** and **outgoing** directories within a new return bay.
- Packets placed in a return bay get their ownership and access mode from the directory itself.
- The **incoming** and **outgoing** directories must be on the same file system.

Default: The default configuration file defines a default return bay, **/usr/atria/shipping/ms_rtn**. The default bay has subdirectories named **incoming** and **outgoing**, and shipping operations look for packets in these subdirectories.

Expiration period

EXPIRATION *storage-class number-of-days*

EXPIRATION –default *number-of-days*

Specifies the expiration period (in days) for shipping orders generated in the specified storage class. This period begins at the time the shipping order is generated. If a packet cannot be delivered to all of its specified destinations, the packet is returned to the original sending host and one or more electronic mail messages are sent (see the descriptions in the sections **Administrator E-mail** and **Notification**).

Specifying **–default** as the storage class sets the expiration period for shipping orders that are not assigned to any storage class, and for shipping orders whose storage class is not configured.

A zero **EXPIRATION** value specifies no expiration and delivery is reattempted indefinitely.

This setting is overridden by the **–pexpire** option to **syncreplica** or **mkreplica**.

Retries — Repeated Attempts to Deliver a Packet

The **shipping_server** program does not retry delivery of a packet.

Default: 14 days.

Packet routing

ROUTE *next-hop host* ...

ROUTE *next-hop* **–default**

Controls the network routing of packets. Packets whose final destination is any of the *host* arguments are sent to the host named *next-hop*. This host is responsible for final delivery of the packet to its destinations (or additional forwarding). *next-hop* and *host* can be host names (which must be usable by hosts in different domains) or numeric IP addresses.

You can include multiple **ROUTE** entries in the configuration file. The special keyword **–default** accommodates all hosts that are not specified in another **ROUTE** entry.

Default: None.

Receipt handler

RECEIPT-HANDLER *storage-class script-pathname*

Specifies a script for the **shipping_server** to run for each packet received by a shipping bay. By default, no script is specified.

For each packet that is received, **shipping_server** handles it as follows:

1. Reads the **shipping.conf** file to find the appropriate **RECEIPT-HANDLER** entry for the packet.
   - If the packet is associated with a storage class and there is a **RECEIPT-HANDLER** entry for that storage class, **shipping_server** uses the *script-pathname* specified in that entry.
   - If the packet is not associated with a storage class and there is a **RECEIPT-HANDLER** value for the **–default** storage class, **shipping_server** uses the script specified for **–default**.
2. Invokes the receipt handler as follows:

```
script-pname [ -d·ata packet-file-pname ]
[ -a·ctual shipping-order-pname ] [ -s·class storage-class ]
-o·rigin hostname
```

where

script-pname	Script specified in the **RECEIPT-HANDLER** entry.
–d·ata *packet-file-pname*	Location of the packet. This parameter is used only when the packet is destined for this host.
–a·ctual *shipping-order-pname*	Location of the shipping order. This parameter is used only when the packet is destined for another host.
–s·class *storage-class*	Storage class associated with the packet. This parameter is used only if the packet was associated with a storage class when it was created.
–o·rigin *hostname*	Host name of the machine from which the packet was first sent.

Note: If a packet is destined for both the local host and another host, both the **–data** and **–actual** parameters are used. The packet is imported at the replica on the host, and then forwarded to its next destination.

Default: None.

Port numbers

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

Warning: Set these entries only on hosts that can communicate through the firewall and have been installed with the MultiSite **shipping_server-**only option.

These entries specify the range of ports for shipping_server to use on a firewall system, and they are set as environment variables in the shipping_server environment.

Guidelines for setting the values:

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

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

Note: To use **shipping_server** on a firewall system, you must also set the **CLEARCASE_MIN_PORT** and **CLEARCASE_MAX_PORT** environment variables in the **atria_start** script. For more information, see *Specifying port values* on page 124.

Default: None.

shipping_server

Store-and-forward packet transport server.

Synopsis **shipping_server** [**–sc·lass** *storage-class-name*] { **–pol·l** | *sources* ...}

Description This command processes one or more shipping orders on the local host, causing the associated packets (or other files) to be sent to remote sites.

After delivering the data file specified in a shipping order to all its destinations, **shipping_server** deletes the data file unless one of the destinations is the local host.

Note: When **shipping_server** starts processing a shipping order, it locks the order. This prevents subsequent invocations of **shipping_server** from processing the order.

TCP/IP Connection

To transmit a file, **shipping_server** uses UDP to contact the **albd_server** process on the receiving host, and **albd_server** invokes **shipping_server** in receive mode on the receiving host. If you are sending packets through a firewall, **shipping_server** tries to use TCP to contact the remote **albd_server**. If that connection fails, **shipping_server** uses UDP. For more information, see In*stalling Store-and-Forward on a Firewall Host* on page 95.

On UNIX, **shipping_server** forks one subprocess for each packet that it needs to send. As many as 10 separate **shipping_server** subprocesses, each trying to send a single packet, can be started for each invocation of **shipping_server**. The same number of subprocesses are forked on the receiving machine. As a subprocess finishes, another can be started, but only 10 can be active simultaneously.

After a TCP connection is established between the two **shipping_server** processes, they transfer the file. The receiving **shipping_server** selects a storage bay using the local store-and-forward configuration settings. See the **MultiSite Control Panel** (Windows). If a storage class is assigned multiple storage bays, available disk space determines the selection of a bay.

After a TCP connection is established between the two **shipping_server** processes, they transfer the file. The receiving **shipping_server** selects a storage bay using the local store-and-forward configuration settings. See **shipping.conf** (UNIX) or the **MultiSite Control Panel** (Windows). If a storage class is assigned multiple storage bays, available disk space determines the selection of a bay.

UNIX: The packet file is created with the same owner and group as the storage bay directory, and its access mode is taken from the directory's read and write permissions. (The execute permission and special permissions, if any, are ignored.)

Windows: The packet file inherits permissions from the Windows ACL on the storage bay directory.

Colon Characters in Packet Names

If a packet name contains a colon (**:**), **shipping_server** changes the colon to a period (**.**) during processing. This change allows packets to be delivered to Windows machines, which do not allow colons within file names.

Handling of File Name Conflicts

You can use the **mkorder** and **shipping_server** commands to transmit arbitrary files if the files are located in the same directory as their associated shipping orders. If a file with the same name already exists on the receiving host, the new file is renamed to *filename_1* (if you send another file with the same name, it is renamed to *filename_2*, and so on).

Log File

UNIX: shipping_server writes records of all packets sent and received, along with all errors, to file /var/adm/atria/log/shipping_server_log.

Windows: **shipping_server** writes records of all packets sent and received, notification messages, and all errors to the Windows event viewer. If the Rational Shipping Server was installed from a Rational ClearCase CD, it writes messages to the Event Viewer in addition to writing messages to the file clearcase-home-dir\var\log\shipping_server_log.

Restrictions

The shipping order and the data file it specifies must be located in the same directory.

Options and Arguments

–sc·lass *storage-class-name*

Processes shipping orders for the specified storage class only.

Default: Processes all shipping orders specified or defined on this machine.

–pol·l

Processes shipping orders located in some (if you use **–sclass**) or all MultiSite storage bays defined in the **MultiSite Control Panel** on Windows.

Processes shipping orders located in some (if you use **–sclass**) or all MultiSite storage bays defined in the **shipping.conf** configuration file on UNIX or the **MultiSite Control Panel** on Windows.

Default: None.

SPECIFYING THE OPLOG ENTRIES TO LIST: shipping_server processes only shipping orders whose file names start with the characters "`sh_o_`". If you create shipping orders, name them according to this convention, or omit the **–poll** option and specify the shipping order pathnames.

On UNIX, only shipping order files that you own are processed. (**EXCEPTION**: when *root* runs this program, shipping order files are processed regardless of ownership.)

sources ...

One or more pathnames of files and / or directories. Each file you specify is processed if it contains a valid shipping order. For each directory you specify, **shipping_server** processes some (if you use **–sclass**) or all shipping orders stored in that directory.

Examples

- Process all shipping orders in all MultiSite storage bays.

```
shipping_server –poll
```

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

```
shipping_server
"c:\Program Files\Rational\ClearQuest\var\shipping\ms_
ship"
```

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

See Also

- mkorder, MultiSite Control Panel
- **shipping.conf** (UNIX)

syncreplica

Generates or applies update packets.

Synopsis

Export an update packet:

sync·replica –exp·ort [**–cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
  -u·ser *username* [**–p·assword**] *password* [**–max·size** *max-packet-size*
  [**–lim·it** *num-packets*]] **–wor·kdir** *directory*
  { { **–sh·ip** | **–fsh·ip**}
  [**–sc·lass** *storage-class*] [**–pex·pire** *date*]
   [**–not·ify** *email*] **|** **–out** { *packet-file-pname* | *staging-area-pname* }
  } *replica ...*

Import an update packet:

sync·replica –imp·ort [**-cl·an** *clan-name*] [**–site** *site-name*] **–fam·ily** *family-name*
  -u·ser *username* [**–p·assword**] *password*
  { **–rec·eive** [**–sc·lass** *storage-class*]
  | { *packet-file-pname* | *staging-area-pname* } ...
  }

Description

Synchronization of an existing replica with one or more replicas at other sites is a two-phase process:

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

Step #2 occurs at all sites that receive the packet.

Contents of an update packet:

- All changes that have occurred in the current database replica since the last update generated for the destination replicas. (Changes already sent to the destination replicas are excluded from the packet).
- Changes that have occurred in other replicas, which the current replica has received in previous update packets from those replicas, but has not already passed on to the destination replicas.

In all cases, **syncreplica –export** creates a single logical update packet for use at all the specified destinations; the packet can be used to update *only* those particular replicas.

Notes on the export phase

MultiSite is designed for efficient updating of replicas. **syncreplica –export** attempts to exclude operations that have been sent previously. (However, there is no harm in sending the same update packet multiple times to the same replica; the first operation is imported and subsequent identical operations are ignored.)

Specifying a directory for temporary files

syncreplica –export stores temporary files in the directory you specify with the -workdir option. This directory must not already exist and is deleted after the export packet is created.

Notes on the import phase

An update packet is applied to the appropriate replicas on the host on which you import it. You do not have to specify particular replicas or storage locations.

The import process applies update packets in the correct order; you can specify packets in any order on the command line.

The database replica is not locked for normal database operations during the import phase. But it is locked for all other MultiSite operations.

Skipping packets

syncreplica –import refuses to process an update packet in the following situations:

- The update packet contains changes that depend on other changes that have not yet been applied to this replica. This usually means that an update packet destined for this replica has not been sent or was lost during transport.
- Problems were encountered processing an earlier physical packet in a multiple-part logical packet.

In any of these cases, **syncreplica –import** displays an explanatory message.

Update Failures / Replaying Packets

In some cases, **syncreplica –import** begins to apply operations to a replica, but terminates with an error message. For example, another process locks the database, causing the import to fail. After the database is unlocked, you can have **syncreplica –import** process the entire update packet again.

There is no harm in importing update packets that have already been processed successfully; the same change will not be made twice. Thus, even importing an entire update packet multiple times causes no error and does no harm.

See *Recovering from lost packets on page 94*, for more information on update failures.

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** commands, it is not deleted.

Hooks Firing

ClearQuest hooks do not fire in response to changes made during packet import.

Handling Naming Conflicts

syncreplica resolves naming conflicts among type objects created at different replicas. For more information, *Resolving naming conflicts* on page 95.

Delayed View Updates

syncreplica does not inform any ClearQuest users of the updates to replicas. All active users see updates within a few seconds, through ClearQuest's normal database-polling routines.

Error Handling for Packet-delivery Failures

If a packet cannot be delivered, it is sent through the store-and-forward facility back to the administrator at the site of the originating replica. A mail message is sent to the store-and-forward administrator. This occurs after repeated attempts to deliver the packet have all failed, and the allotted time has expired; it can also occur when the destination host is unknown or a data file does not exist. The store-and-forward configuration settings specify the expiration period and the e-mail address of the administrator.

Options and Arguments

Export Phase

The following sections describe the options and arguments for use with **syncreplica –export**.

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

- **user database family**: Use the name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.
- **replicated schema repository family**: Not applicable. Update packets automatically include updates to the replicated schema repository, if any.

Default: None.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–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

[**–lim·it** *num-packets*]

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.

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 **MultiSite Control Panel** (Windows).
- Packets created with **–out** are no larger than 2 GB.

–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 replica 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.

Default: None. You must specify how the update packets created by **syncreplica –export** are to be stored and / or transmitted to other sites.

–wor·kdir *directory*

Specify a temporary working directory for syncreplica to use. This directory must not already exist and is deleted after the syncreplica export process.

–sc·lass *class-name*

Specifies the storage class of the packet and shipping order. **syncreplica** looks up the storage class in the **MultiSite Control Panel** on Windows to determine the location of the storage bay to use.

Default: If you omit this option, **syncreplica** places the packet in the storage bay location specified for the *cq_default* class **MultiSite Control Panel**.

–out *packet-file-pname*

Places the first update packet in file *packet-file-pname*. Additional physical packets, if any, are placed in files named *packet-file-pname_2, packet-file-pname_3,* and so on.

Places the packets in the directory indicated with *staging-file-name.*

The update packets are not delivered automatically; use an appropriate mechanism (e-mail, ftp, postal service, and so on) to deliver them.

You can create a packet using **–out**, and deliver it using the store-and-forward facility. See the **mkorder** reference page.

staging-area-path

Name a directory name where packet files generated.

–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.

Windows: This option overrides the storage class's Packet Expiration specification in the MultiSite Control Panel. See the **MultiSite Control Panel** reference page for a description of this specification, and of delivery retries in general.

The *date-time* argument can have any of the following formats:

*date***.***time* | *date* | *time*

where:

date	:=	*day-of-week* | *long-date*
time	:=	*h*[*h*]**:***m*[*m*][**:***s*[*s*]] [**UTC** [[**+** | **-**]*h*[*h*][**:***m*[*m*]]]]
day-of-week	:=	**today** |**yesterday** |**Sunday** | ... |**Saturday** |**Sun** | ... |**Sat**
long-date	:=	*d*[*d*]**–***month*[**–**[*yy*]*yy*]
month	:=	**January** |... |**December** |**Jan** |... |**Dec**

Specify the *time* in 24-hour format, relative to the local time zone. If you omit the time, the default value is **00:00:00**. If you omit the *date*, the default value is **today**. If you omit the century, year, or a specific date, the most recent one is used. Specify **UTC** if you want the time to be resolved to the same moment in time regardless of time zone. Use the plus (+) or minus (-) operator to specify a positive or negative offset to the UTC time. If you specify **UTC** without hour or minute offsets, the default setting is Greenwich Mean Time (GMT). (Dates before January 1, 1970 Universal Coordinated Time (UTC) are invalid.)

Examples:

```
22-November-1998
sunday
yesterday.16:00
8-jun
13:00
today
9-Aug.10:00UTC
```

–not·ify *e-mail-address*

The delivery-failure message is sent to the specified e-mail address.

If a failure occurs on a Windows host that does not have e-mail notification enabled, a message appears in the Windows Event Viewer. The message includes the *e-mail-address* value specified with this option and a note requesting that this user be informed of the status of the operation. For information on enabling e-mail notification, see the **MultiSite Control Panel** reference page.

replica...

You can specify one or more destination replicas. Use the site name to refer to a replica. For example, *sitea* indicates that site a will receive the update packet, while *sitea siteb* indicates that both sites a and b will receive the update packet.

Default: None.

Options and Arguments

Import Phase

The following sections describe the options and arguments for use with **syncreplica –import**.

-cl·an *clan-name*

The name of the clan to which the replica belongs. All the replicas associated with a single schema repository form a clan. The clan name is specified when the replica is activated.

Default: The first clan replicated at this site. If there is more than one clan at this site, this argument is required.

–site *site-name*

The site name of the replica to be modified.

Default: Current site. If there is more than replica on this machine, this argument is required.

–fam·ily *family-name*

The name of the family to which this replica belongs. The replicas of a single user database form a family. The family name is five-character logical database name that was given to the user database when it was created within ClearQuest.

–u·ser *user*

The name of a user with the appropriate database privileges. You must have Super User privileges to administer ClearQuest MultiSite.

–p·assword *password*

The password associated with the user account being used to run this command. Note that you must have Super User privileges to administer ClearQuest MultiSite.

–rec·eive [**–sc·lass** *storage-class*]

Scans one or more of the current host's storage bays. Any unprocessed update packets intended for this host are applied to the appropriate replicas on the host. Using the **–sclass** option restricts processing to the storage bays of the specified storage class.

Default: If **-sclass** option is not specified, this command updates all replicas that are on the current host and are specified in the update packets.

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 MultiSite Control Panel, a message appears in the Windows Event Viewer.) Use **mkreplica** to import these replica-creation packets.

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.

Default: There is no default update packet location.

Examples

Exports

- Generate an update packet to be sent to replica **boston**. Store the packet in a file in directory d:\transfer\packet

```
multiutil syncreplica -export -clan telecomm -site
bangalore -family PRODA -user bangaloreadmin -password
secret -maxsize 500mb -workdir c:\work -out
d:\transfer\packet boston
```

- 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.

```
multiutil syncreplica -export -clan telecomm -site
bangalore -family PRODA -user bangaloreadmin -password
secret -maxsize 500mb -workdir c:\work -ship -sclass
cq_default boston
```

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

```
multiutil syncreplica -export -clan telecomm -site
bangalore -family PRODA -user bangaloreadmin -password
secret -maxsize 500mb -workdir c:\work -fship -sclass
cq_default
```

Imports

- Process an incoming update packet in directory /usr/tmp.

```
multiutil syncreplica -import -clan telecomm -site
boston -family PRODA -user bostonadmin -password secret
-receive /usr/tmp
```

- Process all incoming update packets in the specified storage class. If not storage class is specified, the **-receive** option processes all packets associated with the *cq_default* storage class.

```
multiutil syncreplica -import -clan telecomm -site
boston -family PRODA -user bostonadmin -password secret
-receive -sclass cq_default
```

See Also

- **mkorder**
- **mkreplica**
- **MultiSite Control Panel**
- **shipping.conf**(UNIX)

Index