Guide to IBM Informix Enterprise Replication

Table of Contents

Introduction

Section I   Introducing Enterprise Replication

Section II   Setting Up and Managing Enterprise Replication

Chapter 7   **Creating and Managing Replicate Sets**

Chapter 8   **Monitoring and Troubleshooting Enterprise Replication**

Section III   Appendixes

Appendix A   **Command-Line Utility Reference**

Appendix B   **Configuration Parameter Reference**

Appendix C   **onstat Command Reference**

Appendix D   **SMI Table Reference**

Appendix E   **Replication Examples**

Appendix F   **SQLHOSTS Registry Key**

               Index

Introduction

In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual describes Informix Enterprise Replication and the concepts of data replication. This manual explains how to design your replication system, as well as administer and manage data replication throughout your enterprise.

This section discusses the intended audience and the associated software products that you must have to use Enterprise Replication.

Types of Users

This manual is for database server administrators and assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with database server administration, operating-system administration, and network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to your *Getting Started* manual for a list of supplementary titles.

Software Dependencies

To use Enterprise Replication, you must use Informix Dynamic Server as your database server. Check the release notes for specific version compatibility.

This manual assumes that you are using Dynamic Server, Version 9.3, as your database server.

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All the information related to character set, collation and representation of numeric data, currency, date, and time is brought together in a single environment, called a GLS (Global Language Support) locale.

The examples in this manual are written with the assumption that you are using the default locale, **en_us.8859-1**. This locale supports U.S. English format conventions for date, time, and currency. In addition, this locale supports the ISO 8859-1 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

Demonstration Databases

The DB-Access utility, which is provided with your Informix database server products, includes one or more of the following demonstration databases:

- The **stores_demo** database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. Many examples in Informix manuals are based on the **stores_demo** database.

- The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines. ♦

For information about how to create and populate the demonstration databases, see the *DB-Access User's Manual*. For descriptions of the databases and their contents, see the *Informix Guide to SQL: Reference.*

The scripts that you use to install the demonstration databases reside in the **$INFORMIXDIR/bin** directory on UNIX platforms and in the **%INFORMIXDIR%\bin** directory in Windows environments.

New Features in Dynamic Server, Version 9.3

This section describes new features relevant to this manual. These features fall into the following areas:

- Extensibility Enhancements
- Performance Improvements
- Functionality Enhancements
- Command-Line Changes

For a description of all new features, see the *Getting Started* manual.

Extensibility Enhancements

Enterprise Replication in Version 9.3 of Dynamic Server provides support for replicating the following extensible data types:

- Data stored as smart large objects (smart blobs) in sbspaces (CLOB and BLOB data types), CLOB and BLOB columns (explicitly specified in the table schema), and updates to CLOB and BLOB columns (with some restrictions)

  See "Replicating Simple and Smart Large Objects" on page 2-20.

- Opaque user-defined types (UDTs)

  Version 9.3 does not include support for replication of the following user-defined types:

  - ❏ Row types
  - ❏ Collections
  - ❏ Lists
  - ❏ Sets and multisets

  For more information on which user-defined types are not supported, see the release notes.

- Informix Spatial DataBlade module

In addition, this release of Enterprise Replication allows the following (with some restrictions):

- UDT column references and UDRs in replicate WHERE clauses
- UDTs for primary key columns

For more information about restrictions on replicating opaque types, see "Considerations for Replicating Opaque Data Types" on page 2-25.

- **Support Functions**

  To replicate UDTs, Enterprise Replication requires that the UDT designer provide two support functions: **streamwrite( )** and **streamread( )**. See "UDT Support Functions" on page 2-25. For information on writing these support functions, see the section on Enterprise Replication stream support functions in the *DataBlade API Programmer's Manual*.

Performance Improvements

Enterprise Replication for Dynamic Server, Version 9.3 includes the following performance improvements by improving parallel processing:

- Enterprise Replication now applies all replicates (in replicate sets and individually) in parallel by default.
- Enterprise Replication threads now apply transactions from the same source in parallel unless they contain updates to the same row.
- Enterprise Replication threads normally commit on the target in the same order as on the source.
- Enterprise Replication threads can commit out of order on the target if there are no conflicts.
- Enterprise Replication now uses buffered logging to apply transactions.

Improved parallel processing is built-in and requires no user configuration or interaction. However, this feature is automatically disabled if you are using page-level locking.

Functionality Enhancements

Enterprise Replication for Dynamic Server, Version 9.3 includes the following functionality enhancements:

■ SERIAL Column Primary Keys

The CDR_SERIAL configuration parameter enables control over generating values for **SERIAL** and **SERIAL8** columns in tables defined for replication. This is useful for generating **SERIAL** column primary keys in an Enterprise Replication environment.

For more information, see "CDR_SERIAL" on page B-11.

■ Replicate Sets

You can now manage replicates together in *replicate sets,* both individually and as part of the set.

For more information, see Chapter 7, "Creating and Managing Replicate Sets."

Pre-Version 9.3 replicate groups are not supported in Version 9.3 and later. Before migrating to Version 9.3, you must remove any replicate groups. Follow the important instructions in the *Informix Migration Guide.*

■ Replicating Only Changed Columns

This release of Enterprise Replication provides the ability to replicate only the columns that changed, rather than the entire row. For more information, see "Replicating Only Changed Columns" on page 5-11.

If only changed columns are replicated, the data for all replicated columns might not be available for spooling to the ATS (Aborted Transaction Spooling) and RIS (Row Information Spooling) files. Therefore, the format for these files has changed. For more information, see "Aborted Transaction Spooling Files" on page 8-3 and "Row Information Spooling Files" on page 8-9.

■ Spooling Changes

Enterprise Replication now spools row data in the send and receive queues to an sbspace, specified by the CDR_QDATA_SBSPACE configuration parameter (see "CDR_QDATA_SBSPACE" on page B-9). In addition, you can control logging of these sbspaces. For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-11.

Enterprise Replication also now spools transaction records from the send and receive queues to a single dbspace, specified by the CDR_QHDR_DBSPACE configuration parameter (see "CDR_QHDR_DBSPACE" on page B-8).

For more information, see "Defining Replication Servers" on page 5-5.

■ In-place ALTER Statement

The following ALTER statements are now processed as in-place alters (in most cases) and therefore complete much more quickly:

```
ALTER TABLE ... ADD CRCOLS
ALTER TABLE ... DROP CRCOLS
```

The in-place processing of these ALTER statements is implicit and requires no user action.

For more information about in-place alters, see the "Altering a Table Definition" section in the *Performance Guide*. For more information about CRCOLS, see "Preparing Tables for Conflict Resolution" on page 4-21.

Command-Line Changes

This release includes updates to the command-line interface to support new features, including a new **cdr finderr** utility that looks up a specific Enterprise Replication error number and displays the corresponding error text.

For more information, see Appendix A, "Command-Line Utility Reference."

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
italics **italics** `italics`	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface **boldface**	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
`monospace` `monospace`	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
♦	This symbol indicates the end of one or more product- or platform-specific paragraphs.
→	This symbol indicates a menu item. For example, "Choose **Tools→Options**" means choose the **Options** item from the **Tools** menu.

Tip: When you are instructed to "enter" characters or to "execute" a command, immediately press RETURN *after the entry. When you are instructed to "type" the text or to "press" other keys, no* RETURN *is required.*

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

Comment Icons

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
	Warning:	Identifies paragraphs that contain vital instructions, cautions, or critical information
	Important:	Identifies paragraphs that contain significant information about the feature or operation that is being described
	Tip:	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Feature, Product, and Platform Icons

Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
GLS	Identifies information that relates to the Informix Global Language Support (GLS) feature
UNIX	Identifies information that is specific to UNIX operating system
WIN NT/2000	Identifies information that is specific to the Windows NT and Windows 2000 environments

These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the information that applies ends at the next heading at the same or higher level. A ♦ symbol indicates the end of information that appears in one or more paragraphs within a section.

Command-Line Conventions

This section defines and illustrates the format of commands that are available in Informix products. These commands have their own conventions, which might include alternative forms of a command, required and optional parts of the command, and so forth.

Each diagram displays the sequences of required and optional elements that are valid in a command. A diagram begins at the upper-left corner with a command. It ends at the upper-right corner with a vertical line. Between these points, you can trace any path that does not stop or back up. Each path describes a valid form of the command. You must supply a value for words that are in italics.

You might encounter one or more of the following elements on a command-line path.

Element	Description
command	This required element is usually the product name or other short word that invokes the product or calls the compiler or preprocessor script for a compiled Informix product. It might appear alone or precede one or more options. You must spell a command exactly as shown and use lowercase letters.
variable	A word in italics represents a value that you must supply, such as a database, file, or program name. A table following the diagram explains the value.
-flag	A flag is usually an abbreviation for a function, menu, or option name, or for a compiler or preprocessor argument. You must enter a flag exactly as shown, including the preceding hyphen.

(1 of 2)

Element	Description
.ext	A filename extension, such as **.sql** or **.cob**, might follow a variable that represents a filename. Type this extension exactly as shown, immediately after the name of the file. The extension might be optional in certain products.
(. , ; + * - /)	Punctuation and mathematical notations are literal symbols that you must enter exactly as shown.
' '	Single quotes are literal symbols that you must enter as shown.
Privileges p. 5-17 Privileges	A reference in a box represents a subdiagram. Imagine that the subdiagram is spliced into the main diagram at this point. When a page number is not specified, the subdiagram appears on the same page.
— ALL —	A shaded option is the default action.
⟶	Syntax within a pair of arrows indicates a subdiagram.
—⊣	The vertical line terminates the command.
-f — OFF — / ON	A branch below the main path indicates an optional path. (Any term on the main path is required, unless a branch can circumvent it.)
, / variable	A loop indicates a path that you can repeat. Punctuation along the top of the loop indicates the separator symbol for list items.
, / $\overline{3}$ size	A gate ($\overline{3}$) on a path indicates that you can only use that path the indicated number of times, even if it is part of a larger loop. You can specify *size* no more than three times within this statement segment.

(2 of 2)

How to Read a Command-Line Diagram

Figure 1 shows a command-line diagram that uses some of the elements that are listed in the previous table.

Figure 1
Example of a Command-Line Diagram

```
setenv ──────── INFORMIXC ──────────── compiler ─────────────────────┤
                                    └─── pathname ───┘
```
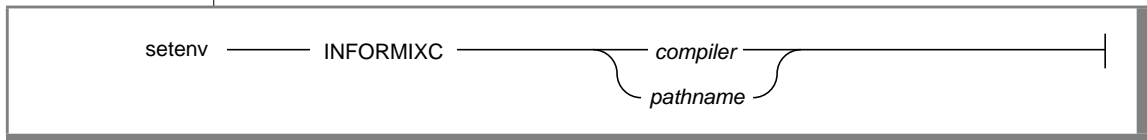
To construct a command correctly, start at the top left with the command. Follow the diagram to the right, including the elements that you want. The elements in the diagram are case sensitive.

Figure 1 illustrates the following steps:

1. Type `setenv`.
2. Type `INFORMIXC`.
3. Supply either a compiler name or a pathname.

    After you choose *compiler* or *pathname*, you come to the terminator. Your command is complete.
4. Press RETURN to execute the command.

Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single Informix application development tool. If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using DB-Access, you must delimit multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the manual for your product.

Additional Documentation

Informix Dynamic Server documentation is provided in a variety of formats:

- **Online manuals.** The Informix OnLine Documentation Web site at http://www.informix.com/answers contains manuals that Informix provides for your use. This Web site enables you to print chapters or entire books.

- **Online help.** Informix provides online help with each graphical user interface (GUI) that displays information about those interfaces and the functions that they perform. Use the help facilities that each GUI provides to display the online help.

  This facility can provide context-sensitive help, an error message reference, language syntax, and more. To order a printed manual, call 1-800-331-1763 or send email to moreinfo@informix.com. Provide the following information when you place your order:

  - ❑ The documentation that you need
  - ❑ The quantity that you need
  - ❑ Your name, address, and telephone number

- **Documentation notes.** Documentation notes, which contain additions and corrections to the manuals, are also located at the OnLine Documentation site at http://www.informix.com/answers. Examine these files before you begin using your database server.

■ **Release notes.** Release notes contain vital information about application and performance issues. These files are located at http://www.informix.com/informix/services/techinfo. This site is a password controlled site. Examine these files before you begin using your database server.

Documentation notes, release notes, and machine notes are also located in the directory where the product is installed. The following table describes these files.

UNIX

On UNIX platforms, the following online files appear in the **$INFORMIXDIR/release/en_us/0333** directory.

Online File	Purpose
erep_docnotes_9.30.html	The documentation notes file for your version of this manual describes topics that are not covered in the manual or that were modified since publication.
release_notes_9.30.html	The release notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
machine_notes_9.30.txt	The machine notes file describes any special actions that you must take to configure and use Informix products on your computer. Machine notes are named for the product described.

♦

Windows

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix Dynamic Server 9.30→ Documentation Notes or Release Notes** from the task bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals with information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

Machine notes do not apply to Windows NT platforms. ♦

■ **Error message files**. Informix software products provide ASCII files that contain Informix error messages and their corrective actions. For a detailed description of these error messages, refer to *Informix Error Messages* in Answers OnLine.

UNIX

To read the error messages on UNIX, use the following command.

Command	Description
finderr	Displays error messages online

♦

WIN NT

To read error messages and corrective actions on Windows NT, use the **Informix Find Error** utility. To display this utility, choose **Start→Programs→Informix** from the task bar. ♦

Related Reading

For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your *Getting Started* manual.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

We want to know about any corrections or clarifications that you would find useful in our manuals that would help us with future versions. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Send electronic mail to us at the following address:

doc@informix.com

This address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact Informix Customer Services.

We appreciate your suggestions.

Introducing Enterprise Replication

Section I

About Informix Enterprise Replication

In This Chapter

Data replication generates and manages multiple copies of data at one or more sites, which allows an enterprise to share corporate data throughout its organization.

This chapter introduces Informix Enterprise Replication and explains how this product replicates data.

Informix Enterprise Replication

Informix Enterprise Replication is an asynchronous, log-based tool for replicating data between Informix Dynamic Server database servers. Enterprise Replication on the source server captures transactions to be replicated by reading the logical log, storing the transactions, and reliably transmitting each transaction as replication data to the target servers.

At each target server, Enterprise Replication receives and applies each transaction contained in the replication data to the appropriate databases and tables as a normal, logged transaction.

Informix Enterprise Replication provides the following:

- Asynchronous Data Replication
- Log-Based Data Capture
- High Performance
- High Availability
- Consistent Information Delivery
- Flexible Architecture
- Centralized Administration

Asynchronous Data Replication

Enterprise Replication uses *asynchronous* data replication to update the databases that reside at a replicated site after the primary database has committed a change.

With asynchronous replication, the delay to update the replicated-site databases can vary depending on the business application and user requirements. However, the data eventually synchronizes to the same value at all sites. The major benefit of this type of data replication is that if a particular database server fails, the replication process can continue and all transactions in the replication system will be committed.

In contrast with *synchronous* data replication, the replicated data is updated immediately when the source data is updated. Synchronous data replication uses the *two-phase commit* technology to protect data integrity. In a two-phase commit, a transaction is applied only if *all* interconnected distributed sites agree to accept the transaction. Synchronous data replication is appropriate for applications that require immediate data synchronization. However, synchronous data replication requires that all hardware components and networks in the replication system be available at all times. For more information about synchronous replication, refer to the discussion of two-phase commit in your *Administrator's Guide for Informix Dynamic Server*.

Asynchronous replication is the more common type of data replication in open-system environments because it protects against system and network failures.

Asynchronous replication allows the following replication models:

- Primary-target ("Primary-Target Replication System" on page 3-3)

  All database changes originate at the primary database and are replicated to the target databases. Changes at the target databases are not replicated to the primary.

- Update-anywhere ("Update-Anywhere Replication System" on page 3-10)

  All databases have read and write capabilities. Updates are applied at all databases.

The update-anywhere model provides the greater challenge in asynchronous replication. For example, if a replication system contains three replication sites that all have read and write capabilities, conflicts occur when the sites try to update the same data at the same time. Conflicts must be detected and resolved so that the data elements eventually have the same value at every site. For more information, see "Conflict Resolution" on page 3-11.

Log-Based Data Capture

Enterprise Replication uses *log-based data capture* to gather data for replication. Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication and then evaluates the row images.

Log-based data capture takes changes from the logical log and does not compete with transactions for access to production tables. Log-based data-capture systems operate as part of the normal database-logging process and thus add minimal overhead to the system.

Two additional methods of data capture, which Enterprise Replication does not support, include:

- Trigger-based data capture

  A trigger is code in the database that is associated with a piece of data. When the data changes, the trigger activates the replication process.

- Trigger-based transaction capture

  A trigger is associated with a table. Data changes are grouped into transactions and a single transaction might trigger several replications if it modifies several tables. The trigger receives the whole transaction, but the procedure that captures the data runs as a part of the original transaction, thus slowing down the original transaction.

High Performance

Enterprise Replication provides high performance by not overly burdening the source of the data and by using networks and all other resources efficiently.

Because Enterprise Replication captures changes from the logical log instead of competing with transactions that access production tables, Enterprise Replication minimizes the effect on transaction performance. Because the capture mechanism is internal to the database, the database server implements this capture mechanism efficiently. For more information, see "Log-Based Data Capture" on page 1-5.

All Enterprise Replication operations are performed in parallel, which further extends the performance of Enterprise Replication. For more information, see "Performance Improvements" on page 7 of the Introduction.

High Availability

Because Enterprise Replication implements asynchronous data replication, network and target database server outages are tolerated. In the event of a database server or network failure, the local database server continues to service local users. The local database server stores replicated transactions in persistent storage until the remote server becomes available.

Consistent Information Delivery

Enterprise Replication protects data integrity. All Enterprise Replication transactions are stored in a reliable queue to maintain the consistency of transactions.

Enterprise Replication uses a data-synchronization process to ensure that transactions are applied at the target database servers in any order equivalent to the order that they were committed on the source database server. If Enterprise Replication can preserve the consistency of the database, Enterprise Replication might commit transactions out-of-order on the target database.

If update conflicts occur, Enterprise Replication provides built-in automatic conflict detection and resolution. You can configure the way conflict resolution behaves to meet the needs of your enterprise. For more information, see "Conflict Resolution" on page 3-11.

Flexible Architecture

Enterprise Replication allows replications based on specific business and application requirements and does not impose model or methodology restrictions on the enterprise.

Enterprise Replication supports both primary-target and update-anywhere replication models. For more information, see "Selecting the Enterprise Replication System" on page 3-3.

Enterprise Replication supports the following network topologies:

- Fully connected

   Continuous connectivity between all participating database servers.

- Hierarchical tree

   A parent-child configuration that supports continuous and intermittent connectivity.

- Forest of trees

   Multiple hierarchical trees that connect at the root database servers.

For more information, see "Choosing a Replication Network Topology" on page 3-19.

Enterprise Replication supports all built-in Informix data types, as well as opaque user-defined data types. For more information, see "New Features in Dynamic Server, Version 9.3" on page 5 of the Introduction and "Enterprise Replication Data Types" on page 2-19.

Enterprise Replication operates in a LAN, WAN, and combined LAN/WAN configuration across a range of network transport protocols.

Enterprise Replication supports the Global Language Support (GLS) feature, which allows Informix products to handle different languages, regional conventions, and code sets. For more information, see "Using GLS with Enterprise Replication" on page 2-18.

Centralized Administration

Enterprise Replication allows administrators to easily manage all the distributed components of the replication system from a single point of control.

You can use the command-line utility (CLU) to administer the replication system from your system command prompt and connect to other servers involved in replication, as necessary. For information, see Appendix A, "Command-Line Utility Reference."

In addition, you can use Informix Server Administrator (ISA) to administer your replication system from a web browser. For more information, see the Documentation Notes described in "Additional Documentation" on page 15 of the Introduction.

How Enterprise Replication Replicates Data

Before you can replicate data, you must declare a database server for replication and define the *replicates* (the data to replicate and the database servers that participate in replication). To declare a database server for replication, see "Defining Replication Servers" on page 5-5. To define replicates, see "Defining Replicates" on page 5-7. Appendix E, "Replication Examples," has simple examples of declaring replication servers and defining replicates.

After the servers and replicates are defined, Enterprise Replication uses the following steps to replicate data:

1. Capture Transactions.
2. Evaluate Data for Replication.
3. Distribute Data.
4. Apply Replicated Data.

Capture Transactions

As the database server writes rows to the logical log, it marks rows that should be replicated. Later, Enterprise Replication reads the logical log to obtain the row images for tables that participate in replication.

Informix database servers manage the logical log in a circular fashion; the most recent logical-log entries write over the oldest entries. Enterprise Replication must read the logical log quickly enough to prevent new logical-log entries from overwriting the logs Enterprise Replication has not yet processed. If the database server comes close to overwriting a logical log that Enterprise Replication has not yet processed, user transactions are blocked until Enterprise Replication can advance. (This situation is called *blockout mode* and occurs only if the system is severely misconfigured.)

The row images that participate in replication are passed to Enterprise Replication for further evaluation.

Evaluate Data for Replication

Enterprise Replication evaluates transactions based on a change to a final image of a row.

Row Images

Enterprise Replication evaluates the initial and final images of a row and any changes that occur between the two row images to determine which rows to replicate. Each row image contains the data in the row as well as the action performed on that row.

A row might change more than once in a particular transaction. For example, a transaction might insert and then update a row prior to committing. Enterprise Replication evaluates the net effect (final state) of a transaction based on the row buffers in the log. Enterprise Replication then determines what should replicate, based on the net effect, the initial state of the row, and whether the replicate definition (in particular, the WHERE clause) applies to the initial and final state.

The table in Figure 1-1 shows the logic that determines which rows are candidates for replication.

Evaluate Data for Replication

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server	Comments
INSERT	T *or* F	DELETE	T *or* F	Yes *or* no	Nothing	Net change of transaction results in no replication
INSERT	T *or* F	UPDATE	T	Yes *or* no	INSERT with final row image	Inserts final data of transaction
INSERT	T *or* F	UPDATE	F	Yes *or* no	Nothing	Final evaluation determines no replication
UPDATE	T	DELETE	T *or* F	Yes *or* no	DELETE with initial row image	Net result of transaction is delete
UPDATE	F	DELETE	T *or* F	Yes *or* no	Nothing	Net change of transaction results in no replication
UPDATE	T	UPDATE	T	Yes	DELETE with initial row image and INSERT with final row image	Ensures old primary key does not replicate
UPDATE	T	UPDATE	T	No	UPDATE with final row image	Simple update
UPDATE	T	UPDATE	F	Yes *or* no	DELETE with initial row image	Row no longer matches replicate definition
UPDATE	F	UPDATE	T	Yes *or* no	INSERT with final row image	Row now matches replicate definition
UPDATE	F	UPDATE	F	Yes *or* no	Nothing	No match exists, and therefore, no replication

Where:

- ■ Initial image is the before image of the transaction in the logical log.
- ■ The replicate evaluates to T (true) or F (false).
- ■ Final Image is the image of the transaction that is replicated.

The table in Figure 1-1 on page 1-10 illustrates how Enterprise Replication evaluates the row-image type (INSERT, UPDATE, DELETE), the results of evaluating the replicate WHERE clause for both the initial and final image, and whether the primary key changes as a result of the transaction.

Tip: The evaluation logic in Figure 1-1 on page 1-10 assumes that the source and the destination tables are initially synchronized (identical before replication begins). If the tables are not synchronized, anomalous behavior could result.

After Enterprise Replication identifies transactions that qualify for replication, Enterprise Replication transfers the transaction data to a queue.

Evaluating Rows for Updates

Enterprise Replication evaluates rows for primary-key updates, for WHERE clause column updates, and for multiple updates to the same row. The following list describes an occurrence in a transaction and the Enterprise Replication action:

- Primary-key updates

  Enterprise Replication translates an update of a primary key into a delete of the original row and an insert of the row image with the new primary key. If triggers are enabled on the target system, insert triggers are executed.

- WHERE clause column updates

  If a replicate includes a WHERE clause in its data selection, the WHERE clause imposes selection criteria for rows in the replicated table.

  ❑ If an update changes a row so that it no longer passes the selection criteria on the source, it is deleted from the target table. Enterprise Replication translates the update into a delete and sends it to the target.

  ❑ If an update changes a row so that it passes the selection criteria on the source, it is inserted into the target table. Enterprise Replication translates the update into an insert and sends it to the target.

- Multiple-row images in a transaction

  Enterprise Replication compresses multiple-row images and only sends the net change to the target. Because of this, triggers might not execute on the target database. For more information, see "Triggers" on page 2-13.

Enterprise Replication supports the replication of BYTE and TEXT data types (simple large objects) and BLOB and CLOB data types (smart large objects), and opaque user-defined data types, as well as all built-in Informix data types. However, Enterprise Replication implements the replication of these types of data somewhat differently from the replication of other data types. For more information, see "Extensibility Enhancements" on page 6 of the Introduction, "Replicating Simple and Smart Large Objects" on page 2-20, and "Replicating Opaque User-Defined Data Types" on page 2-25.

Send Data Queues and Receive Data Queues

Enterprise Replication uses send and receive queues to receive and deliver replication data to and from database servers that participate in a replicate:

■ Send queue

Enterprise Replication stores replication data in memory to be delivered to target database servers that participate in a replicate. If the send queue fills, Enterprise Replication spools the send queue transaction records to a dbspace and the send queue row data to an sbspace.

■ Receive queue

Enterprise Replication stores replication data in memory at the target database server until the target database server acknowledges receipt of the data. If the receive queue fills as a result of a large transaction ("Large Transactions" on page 2-15), Enterprise Replication spools the receive queue transaction header to a dbspace and the receive queue row data to an sbspace.

For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-11 and "Preventing Memory Queues from Overflowing" on page 8-13.

The data contains the filtered log records for a single transaction. Enterprise Replication stores the replication data in a stable (recoverable) send queue on the source database server. Target sites acknowledge receipt of data when it is applied to or rejected from the target database.

If a target database server is unreachable, the replication data remains in a stable queue at the source database server. Temporary failures are common and no immediate action is taken by the source database server; it continues to queue transactions. When the target database server becomes available again, queued transactions are transmitted and applied (see "Apply Replicated Data" on page 1-17).

If the target database server is unavailable for an extended period, the send queue on the source database server might consume excessive resources. In this situation, you might not want to save all transactions for the target database server. To prevent unlimited transaction accumulation, you can remove the unavailable target database server from the replicate (see "Managing Replication Servers" on page 6-3). Before the database server that is removed rejoins any replicate, however, you must synchronize (bring tables to consistency) with the other database servers (see "Resynchronizing Replication Servers" on page 5-17).

Data Evaluation Examples

Figure 1-2, Figure 1-4 on page 1-15, and Figure 1-6 on page 1-16 show three examples of how Enterprise Replication uses logic to evaluate transactions for potential replication.
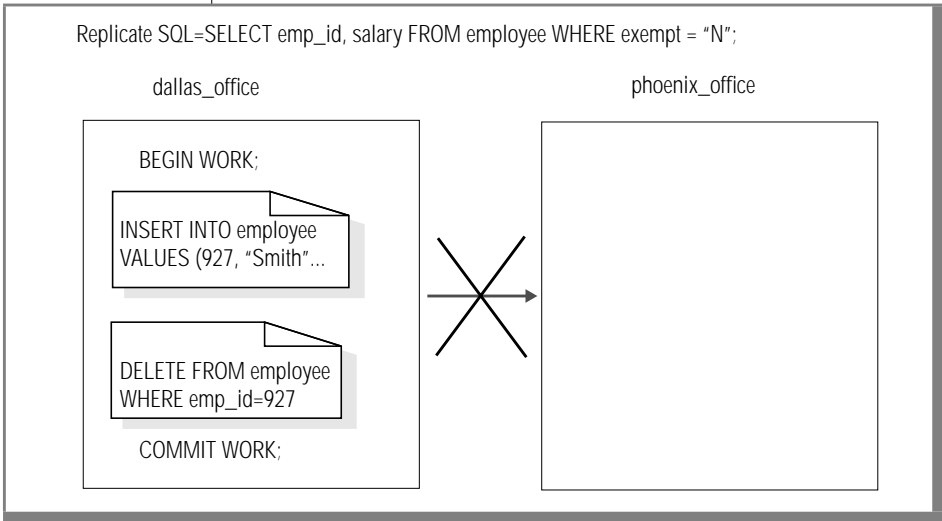
Figure 1-2
Insert Followed by a Delete

Figure 1-2 shows a transaction that takes place at the Dallas office. Enterprise Replication uses the logic in Figure 1-3 to evaluate whether any information is sent to the destination database server at the Phoenix office.

Figure 1-3
Insert Followed by a Delete Evaluation Logic

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
INSERT	T *or* F	DELETE	T *or* F	Yes *or* no	Nothing

Enterprise Replication determines that the insert followed by a delete results in no replication operation; therefore, no replication data is sent.

In Figure 1-4, Enterprise Replication uses the logic in Figure 1-5 to evaluate whether any information is sent to the destination database server.

Figure 1-4
Insert Followed by an Update

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
INSERT	T *or* F	UPDATE	T	Yes or no	INSERT with final row image

The replicate WHERE clause imposes the restriction that only rows with the exempt column equal to "N" are replicated. Enterprise Replication evaluates the transaction (an insert followed by an update) and converts it to an insert to propagate the updated (final) image.

In Figure 1-6, Enterprise Replication uses the logic in Figure 1-7 to evaluate whether any information is sent to the destination database server.

Figure 1-6
Update; Not Selected to Selected

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server
UPDATE	F	UPDATE	T	Yes *or* no	INSERT with final row image

The example shows a replicate WHERE clause column update. A row that does not meet the WHERE clause selection criteria is updated to meet the criteria. Enterprise Replication replicates the updated row image and converts the update to an insert.

Distribute Data

Enterprise Replication ensures that all data reaches the appropriate server, regardless of a network or system failure. In the event of a failure, Enterprise Replication stores data until the network or system is operational. Enterprise Replication replicates data efficiently with a minimum of data copying and sending.

Apply Replicated Data

Enterprise Replication uses a data-synchronization process to apply the replicated data to target database servers. The target database servers acknowledge receipt of data when the data is applied to the target database. The data-synchronization process ensures that transactions are applied at the target database servers in any order equivalent to the order that they were committed on the source database server. If Enterprise Replication can preserve the consistency of the database, Enterprise Replication might commit transactions out of order on the target database.

When Enterprise Replication applies replication data, it checks to make sure that no *collisions* exist. A collision occurs when two database servers update the same data simultaneously. Enterprise Replication reviews the data one row at a time to detect a collision.

If Enterprise Replication finds a collision, it must resolve the conflict before applying the replication data to the target database server.
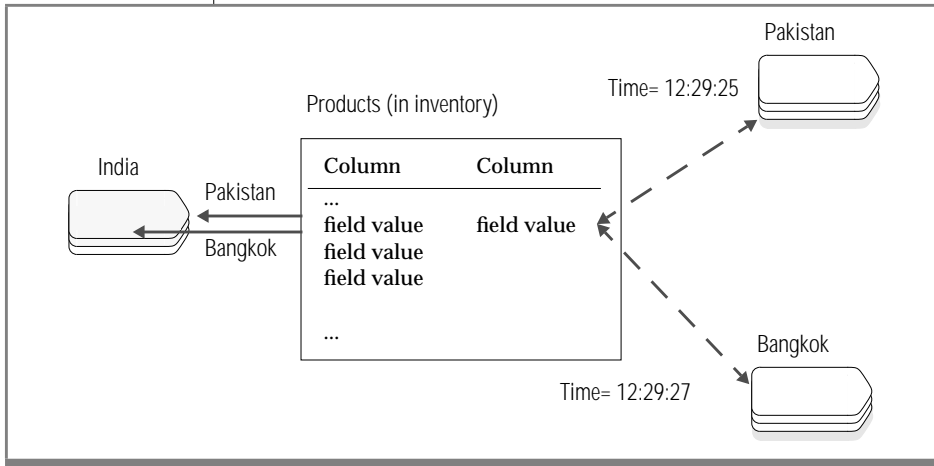
Figure 1-8
Collision Example

Figure 1-8 shows a situation that yields a conflict. Pakistan updates the row two seconds before Bangkok updates the same row. The Bangkok update arrives at the India site first, and the Pakistan update follows. The Pakistan time is earlier than the Bangkok time. Because both updates involve the same data and a time discrepancy exists, Enterprise Replication detects a collision.

For more information, see "Conflict Resolution" on page 3-11.

Enterprise Replication scans to see if the same primary key already exists in the target table or in the associated *delete table*, or if a *replication order error* is detected. A delete table stores the row images of deleted rows. A replication order error is the result of replication data that arrives from different database servers with one of the following illogical results:

- A replicated DELETE that finds no row to DELETE on the target
- An UPDATE that finds no row to UPDATE on the target
- An INSERT that finds a row that already exists on the target

Overview of Enterprise Replication Administration

In This Chapter

This chapter introduces you to Enterprise Replication administration and describes the Enterprise Replication server administrator, terminology, and considerations for using Enterprise Replication.

Overview of Enterprise Replication Administration

To set up Enterprise Replication

1. Select the Enterprise Replication system and network topology to use for your replication environment.

    For information, see Chapter 3, "Selecting the Enterprise Replication System and Network Topology."

2. Prepare the replication environment.

    For information, see Chapter 4, "Preparing the Replication Environment."

3. Initialize the database server.

    For information, see Chapter 5, "Defining and Modifying Replication Servers, Replicates, and Participants."

4. Define database servers for replication.

    For information, see Chapter 5.

5. Define replicates and participants.

    For information, see Chapter 6, "Managing Replication Servers and Replicates."

6. Create replicate sets (optional).

    For information, see Chapter 7, "Creating and Managing Replicate Sets."

7. Start the replicate.

   For information, see Chapter 6.

Once you configure Enterprise Replication, use this information to manage your replication environment:

- "Managing Replication Servers" on page 6-3
- "Managing Replicates" on page 6-7
- "Managing Replicate Sets" on page 7-6
- "Monitoring and Troubleshooting Enterprise Replication" on page 8-1

Enterprise Replication Server Administrator

The Enterprise Replication server administrator must have Informix administrative privileges to configure and manage Enterprise Replication.

Operating System	Privileges
UNIX	user **informix**
Windows	member of the **Informix**-**Admin** group

Enterprise Replication Terminology

The following terms define the data in an Enterprise Replication system and how it is treated:

- Enterprise Replication Server
- Replicate
- Participant
- Replicate Set
- Global Catalog

Enterprise Replication Server

An Enterprise Replication server, or *replication server*, is an Informix database server that participates in data replication. The replication server maintains information about the replication environment, which columns should be replicated, and the conditions under which the data should be replicated. This information is stored in a database, **syscdr**, that the database server creates when it is initialized. Multiple database servers can be on the same physical computer, and each database server can participate in Enterprise Replication.

Important: *For each server participating in replication, you must set up a database server group in the SQLHOSTS file (UNIX) or registry (Windows). For more information, see "Setting up Database Server Groups" on page 4-5 and Appendix F, "SQLHOSTS Registry Key."*

Tip: *This manual uses the convention that the name of a database server group is* **g_** *followed by the name of a database server that is in the group; for example,* **g_italy***.*

For more information, see "Defining Replication Servers" on page 5-5 and "cdr define server" on page A-20.

Replicate

A *replicate* defines the replication *participants* and various attributes of how to replicate the data, such as frequency and how to handle any conflicts during replication.

For more information, see "Defining Replicates" on page 5-7 and "cdr define replicate" on page A-10.

Participant

A *participant* specifies the data (database, table, and columns) to replicate and the database servers to which the data replicates.

Important: *You cannot start and stop replicates that have no participants.*

For more information, see "Defining Participants" on page 5-8 and "Participant" on page A-88.

Replicate Set

A *replicate set* combines several replicates to form a set that can be administered together as a unit.

If your replication system contains many replicates that you define as part of a replicate set, you can use a single command to start, stop, suspend, or resume all the replicates in the set.

For more information, see Chapter 7, "Creating and Managing Replicate Sets" and "cdr change replicateset" on page A-6.

Global Catalog

Each database server that participates in Enterprise Replication maintains tables in the **syscdr** database to keep track of Enterprise Replication configuration information and state. For all *root* and *nonroot* ("Replication Topology Terms" on page 3-21) replication servers, this catalog is a *global catalog* that maintains a global inventory of Enterprise Replication configuration information. The global catalog is created when you define the server for replication.

The global catalog includes the following:

- Enterprise Replication server definitions and state
- Routing and connectivity information
- Replicate definitions and state
- Participant definitions and state
- Replicate set definitions and state
- Conflict detection and resolution rules and any associated SPL routines

The tables in one global catalog instance are automatically replicated to the global catalogs of all other replication servers (except leaf servers), thus you can manage the entire replication environment from one non-leaf replication server. For information about managing replication servers (and their global catalogs), refer to "Managing Replication Servers" on page 6-3.

Leaf replication servers ("Replication Topology Terms" on page 3-21) have a limited catalog. Because the parent database server always manages operations that involve a leaf database server, the catalog of the leaf database server contains only enough data to allow it to interact with its parent server. Limiting the catalog of leaf database servers makes the replication system more efficient because the global catalogs do not need to be replicated to the leaf servers.

For information on defining root, nonroot, and leaf servers, see "Customizing the Replication Server Definition" on page 5-6.

Enterprise Replication Considerations

This section discusses items to consider when planning to use Enterprise Replication:

- Operational Considerations
- Backup and Restore Considerations
- Database and Table Design Considerations
- Transaction Processing Considerations
- Replication Environment Considerations
- Enterprise Replication Data Types

Operational Considerations

Enterprise Replication imposes the following operational limitations:

- Enterprise Replication supports replication on Informix Dynamic Server only.
- High-Availability Data Replication (HDR) cannot run concurrently with Enterprise Replication. For more information, see the chapter on High-Availability Data Replication in the *Administrator's Guide for Informix Dynamic Server*.
- Replication is restricted to base tables. That is, you cannot define a replicate on a view or synonym. A *view* is a synthetic table, a synthesis of data that exists in real tables and other views. A *synonym* is an alternative name for a table or a view. For more information on views and synonyms, see the *Informix Guide to Database Design and Implementation*.

Enterprise Replication asynchronously propagates many control operations through the Enterprise Replication network. When you perform administrative functions using Enterprise Replication, the status returning from those operations is indicative only of the success or failure of the operation at the database server to which you are directly connected. The operation might still be propagating through the other Enterprise Replication database servers in the network at that time.

Due to this asynchronous propagation, avoid performing control operations in quick succession that might directly conflict with one another without verifying that the first operation has successfully propagated through the entire enterprise network. Specifically, avoid deleting Enterprise Replication objects such as replicates, replicate sets, and Enterprise Replication servers, and immediately re-creating those objects with the same name. Doing so can cause failures in the Enterprise Replication system at the time of the operation or later. These failures might manifest themselves in ways that do not directly indicate the source of the problem.

If you must delete and re-create a definition, use a different name for the new object (for example, delete replicate **a.001** and recreate it as **a.002**) or wait until the delete action has successfully propagated through the entire Enterprise Replication system before you re-create the object. The former strategy is especially appropriate if you have database servers that are not connected to the Enterprise Replication network at all times. It might take significant time before the operation is propagated to those disconnected servers.

Backup and Restore Considerations

When backing up and restoring database servers that participate in repli-cation, *do not* stop Enterprise Replication before performing a backup on an Enterprise Replication system.

Database and Table Design Considerations

Consider the following when designing databases and tables for replication:

- Unbuffered Logging
- Table Types
- Shadow Columns
- SERIAL Data Types and Primary Keys
- Cascading Deletes
- Triggers

Unbuffered Logging

Databases on all server instances involved in replication must be created with logging.

Informix recommends that you replicate tables only from databases created with unbuffered logging. Enterprise Replication evaluates the logical log for transactions that modify tables defined for replication. If a table defined for replication resides in a database that uses buffered logging, the transactions are not immediately written to the logical log, but are instead buffered and then written to the logical log in a block of logical records. When this occurs, Enterprise Replication evaluates the buffer of logical-log records all at once, which consumes excess CPU time and memory. When you define a table for replication in a database created with unbuffered logging, Enterprise Replication can evaluate the transactions as they are produced.

To create a database with unbuffered logging, use:

```
CREATE DATABASE db_name WITH LOG
```

For more information, see the section on CREATE DATABASE in the *Informix Guide to Database Design and Implementation*.

Table Types

The following table types are not supported by Enterprise Replication:

- RAW tables

  Because RAW tables are not logged, they cannot be replicated using Enterprise Replication.

- Temporary tables

  Because the database server deletes temporary tables when an application terminates or closes the database, you should not include these tables in your replication environment.

For more information on table types, see *Informix Guide to Database Design and Implementation*.

Out-of-Row Data

Enterprise Replication collects out-of-row data for transmission after the user transaction has committed. Due to activity on the replicated row, the data might not exist at the time Enterprise Replication collects it for replication. In such cases, Enterprise Replication normally applies a NULL on the target system. Therefore, you should avoid placing a NOT NULL constraint on any replicated column that includes out-of-row data.

Shadow Columns

In an update-anywhere replication environment, you must provide for conflict resolution using a conflict-resolution rule (see "Conflict Resolution" on page 3-11). When you create a table that uses the time stamp or time stamp plus SPL conflict-resolution rule, you must define the shadow columns, **cdrserver** and **cdrtime** on both the source and target replication servers.

*Tip: If you plan to use only the ignore conflict-resolution rule, you do not need to define the **cdrserver** and **cdrtime** shadow columns.*

For more information, see "Preparing Tables for Conflict Resolution" on page 4-21.

Primary Key Constraint

All tables involved in replication must have a PRIMARY KEY constraint defined on at least one column, which forces the column to be unique. (For more information about primary keys, see the *Informix Guide to Database Design and Implementation* and the *Informix Guide to SQL: Syntax.*)

Important: Because primary key updates are sent as DELETE/INSERT statement pairs, avoid changing the primary key and updating data in the same transaction.

SERIAL Data Types and Primary Keys

If you plan to use SERIAL data types (SERIAL and SERIAL8) as the primary key for a table, the same serial value might be generated on two servers at the same time.

To avoid this problem, use the CDR_SERIAL configuration parameter to generate non-overlapping (unique) values for SERIAL columns across all database servers in your replication environment. Set CDR_SERIAL in the ONCONFIG file for each primary (source) database server. For more information and examples, see "CDR_SERIAL" on page B-11.

If you do not set CDR_SERIAL, you must specify that the serial column is part of a composite primary key, to avoid generating non-unique SERIAL primary keys. The non-serial column part of the primary key identifies the server on which the row was initially created.

Cascading Deletes

If a table includes a *cascading delete*, when a parent row is deleted, the children are also deleted. If both the parent and child tables participate in replication, the deletes for both the parent and child are replicated to the target servers.

If the same table definition exists on the target database, Enterprise Replication attempts to delete the child rows twice. Enterprise Replication usually processes deletes on the parent first and then the children. When Enterprise Replication processes deletes on the children, an error might result, because the rows were already deleted when the parent was deleted. The table in Figure 2-1 indicates how Enterprise Replication resolves cascading deletes with conflict-resolution scopes and rules.

For more information on cascading deletes, see the ON DELETE CASCADE section in the *Informix Guide to SQL: Syntax*.

Figure 2-1
Resolving Cascade Deletes

Conflict-Resolution Rule	Conflict-Resolution Scope	Actions on Delete Errors	Result
Time stamp	Row-by-row or transaction	Note as replication exceptions	Process children rows
Ignore	Transaction	Report as errors	Abort entire transaction
Ignore	Row-by-row	Report as errors	Reject row

Triggers

A *trigger* is a database object that automatically sets off a specified set of SQL statements when a specified event occurs.

If the --**firetrigger** option is enabled on a replicate, any triggers defined on a table that participates in replication are invoked when transactions are processed on the target server. However, because Enterprise Replication only replicates the final result of a transaction, triggers execute only once on the target regardless of how many triggers executed on the source. In cases where the final evaluation of the transaction results in no replication (for example, an INSERT where the final row image is a DELETE, as shown in Figure 1-3 on page 1-15), no triggers execute on the target database.

If the same triggers are defined on both the source and target tables, any INSERT, UPDATE, or DELETE operation that the triggers generate are also sent to the target database server. For example, the target table might receive replicate data caused by a trigger that also executes locally. Depending on the conflict-resolution rule and scope, these operations can result in errors. To avoid this problem, define the replicate to not fire triggers on the target table.

For more information on triggers, see "Enabling Triggers" on page 5-14 and the CREATE TRIGGER section in *Informix Guide to SQL: Syntax*.

Transaction Processing Considerations

Many variables affect the impact that replicating data has on your transaction processing. This section discusses some of these variables:

- Replication Volume
- Distributed Transactions
- Large Transactions
- Supported SQL Statements

Replication Volume

To determine replication volume, you must estimate how many data rows change per day. For example, an application issues a simple INSERT statement that inserts 100 rows. If this table is replicated, Enterprise Replication must propagate and analyze these 100 rows before applying them to the targets.

Distributed Transactions

A *distributed transaction* is a transaction that commits data in a single transaction over two or more database servers.

Outside of the replication environment, Dynamic Server uses a two-phase commit protocol to ensure that the transaction is either committed completely across all servers involved or is not committed on any server. For more information about the two-phase commit protocol, see the *Administrator's Guide for Informix Dynamic Server*.

In a replication environment, when a distributed transaction is committed across the source servers, each part of the transaction that applies to the local server is written to the local logical logs. When Enterprise Replication retrieves the transaction from the logical logs and forms its transaction data, it is unable to identify the separate transaction data as the original single transaction.

This situation could result in Enterprise Replication applying one transaction successfully while aborting another. Another result might be a time lapse between when Enterprise Replication applies the two transactions (depending on the number of transaction data in each server's send queue and the state of the server).

Large Transactions

Because Enterprise Replication is optimized to handle small transactions efficiently, avoid replicating very large transactions or batch jobs on target systems.

Instead of using Enterprise Replication to perform a batch job, Informix recommends that you use the BEGIN WORK WITHOUT REPLICATION to run the batch job locally on each database server. For more information, see "Blocking Replication" on page 4-18.

Supported SQL Statements

After you define Enterprise Replication on a table by including that table as a participant in a replicate, you cannot execute any operation that changes the structure of the table. In addition, you cannot exclusively lock a database that is involved in replication (or perform operations that require an exclusive lock). However, you can exclusively lock a table in a database.

Forbidden SQL Statements

You cannot use the following SQL statements against a table that is included in a replicate:

- DROP TABLE
- RENAME TABLE
- ALTER FRAGMENT

Limited SQL Statements

The following additional limitations also apply to tables defined for replication:

- Do not remove or disable the primary key constraint.
- Do not rename, add, or drop columns.
- Do not modify a column type.
- Do not add or drop rowids.
- Do not add or drop CRCOLS (shadow columns):
  - ALTER TABLE ... ADD CRCOLS
  - ALTER TABLE ... DROP CRCOLS

  For more information about CRCOLS, see "Preparing Tables for Conflict Resolution" on page 4-21.
- Do not modify the primary key columns.

  For example, do not alter the column to add default values or other constraints.

Important: *Because primary key updates are sent as DELETE/INSERT statement pairs, avoid changing the primary key and updating data in the same transaction.*

- Do not create or alter clustered indexes.

Important: *Creating or altering a clustered index on a replicated table is not allowed.*

Tip: *To use the forbidden and limited SQL statements against a table defined for replication, you must first stop (not suspend) the replicate that contains the table, before running the SQL statement. After modifying the table, restart the replicate. For more information, see "Managing Replicates" on page 6-7.*

Permitted SQL Statements

Enterprise Replication permits the following SQL statements with no limitations:

- SET object mode (no disabling of primary key constraint)
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE

- CREATE TRIGGER
- DROP TRIGGER
- CREATE VIEW
- DROP VIEW
- CREATE SYNONYM
- DROP SYNONYM
- ADD INDEX
- DROP INDEX
- ALTER INDEX (except TO CLUSTER)
- ALTER TABLE constraints (except for the primary key)

Replication Environment Considerations

Each replication system that you create affects your environment. Consider the following when setting up your replication environment:

- Time Synchronization
- Using GLS with Enterprise Replication

Time Synchronization

Whenever you use replication that requires time stamp conflict resolution (see "Conflict Resolution" on page 3-11), the operating system times of the database servers that participate in the replicate must be synchronized. All time stamps and internal computations are performed in Greenwich Mean Time (GMT) and have an accuracy of plus or minus one second.

Important: *Enterprise Replication does not manage clock synchronization between database servers that participate in a replicate.*

To synchronize the time on one database server with the time on another database server, use one of the following commands.

Operating System	Command
UNIX	rdate *hostname*
Windows	net time *servername* /set
	net time /domain:*servername* /set

Important: *These commands do not guarantee the times will remain synchronized. Informix recommends that you use a product that supplies a network time protocol to ensure that times remain synchronized. For information on tools for synchronizing the times, refer to your operating system documentation.*

GLS

Using GLS with Enterprise Replication

An Enterprise Replication system can include databases in different locales, with the following restrictions:

- When you define a database server for Enterprise Replication, that server must be running in the English locale.

  In other words, the **syscdr** database on every Enterprise Replication server must be in the English locale.

- You can replicate only between databases that are in the same locale.

- Replicate names can be in the locale of the database.

Code-set conversion with the GLS library requires only those code-set conversion files found in the **$INFORMIXDIR/gls/cv9** directory.

- For U.S. English, locales are handled automatically by the Client SDK/Informix Connnect installation and setup.

- For non-U.S. English locales, you might need to explicitly provide the locale and conversion files.

For information about how to specify a nondefault locale and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

Enterprise Replication Data Types

Enterprise Replication supports Informix-defined data types, including limited support of user-defined data types. This section describes how Enterprise Replication handles special data types:

- Replicating on Heterogeneous Hardware
- Replicating Simple and Smart Large Objects
- Replicating Opaque User-Defined Data Types

For general information on data types, refer to the *Informix Guide to SQL: Reference* and *Informix Guide to Database Design and Implementation*.

Important: *Enterprise Replication does not support replication of simple large objects stored on optical devices.*

Important: *Enterprise Replication does not verify the data types of columns in tables that participate in replication. The replicated column in a table on the source database server must have the same data type as the corresponding column on the target server. The exception to this rule is cross-replication between simple large objects and smart large objects.*

If you use SERIAL or SERIAL8 data types, you must be careful when defining serial columns. For more information, see "SERIAL Data Types and Primary Keys" on page 2-12.

Replicating on Heterogeneous Hardware

Enterprise Replication supports all primitive data types across heterogeneous hardware. If you define a replicate that includes non-primitive data types (for example, BYTE and TEXT data), the application must resolve data-representation issues that are architecture dependent.

If you use floating-point data types with heterogeneous hardware, you might need to use IEEE floating point or canonical format for the data transfers. For more information, see "Using the IEEE Floating Point or Canonical Format" on page 5-13.
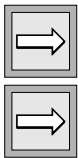
Replicating Simple and Smart Large Objects

Informix supports replicating:

■ Simple large object data types (blobs)

You can store simple large objects either in the tblspace with the rest of the table columns (in a dbspace) or in a blobspace.

■ Smart large object data types (smart blobs)

You must store smart large objects in sbspaces.

For more information about database storage, see the *Administrator's Guide for Informix Dynamic Server*.

Replicating Simple Large Objects from Tblspaces

Simple large object data that is stored in tblspaces (rather than in blobspaces) is placed in the logical log. Enterprise Replication reads the logical log to capture and evaluate the data for potential replication.

Replicating Large Objects from Blobspaces or Sbspaces

Enterprise Replication does not retrieve simple large object data that is stored in blobspaces and smart large object data that is stored in sbspaces from the logical log. Instead, Enterprise Replication retrieves the large object data directly from the blobspace/sbspace before sending the data to the target database server.

It is possible that a transaction subsequent to the transaction being replicated has modified or deleted a simple or smart large object that Enterprise Replication is trying to retrieve.

If Enterprise Replication encounters a row whose large object (simple or smart) has been modified or deleted by a subsequent transaction, Enterprise Replication does not send the data in the large object.

In most cases, the subsequent transaction that modified or deleted the large object will also be replicated, so the data again becomes consistent once that transaction is replicated. The data in the large object is inconsistent only in a small window of time.

Keep in mind that if you specify sending only the columns that changed, the data might not get updated during the next update of the row. For more information, see "Replicating Only Changed Columns" on page 5-11.

Tip: *Enterprise Replication allows cross-replication between simple large objects and smart large objects. For example, you can replicate a simple large object on the source database server to a smart large object on the target server or vice versa.*

Conflict Resolution for Simple and Smart Large Objects

By default, Enterprise Replication performs all conflict detection and resolution at the row level. However, in some cases, simple large object data that is stored in a tblspace (rather than in a blobspace) is accepted by the target server even if the row is rejected. This does not apply to simple large object data that is stored in blobspaces or smart large object data that is stored in sbspaces.

Time-Stamp Conflict Resolution for Simple Large Objects

When a replicated BYTE or TEXT column is modified on the source database server, Enterprise Replication records the value of **cdrserver** and **cdrtime** for that column. (For more information on **cdrserver** and **cdrtime**, see "Preparing Tables for Conflict Resolution" on page 4-21.) If the column on the target database server is also stored in a tablespace (rather than in a blobspace), Enterprise Replication evaluates the **cdrserver** and **cdrtime** values in the source and target columns and uses the following logic to determine if the data is to be applied:

- If the column of the replicated data has a time stamp that is greater than the time stamp of the column on the local row, the data for the column is accepted for replication.

- If the server ID and time stamp of the replicated column are equal to the server ID and time stamp on the column on the local row, the data for the column is accepted for replication.

SPL Conflict Resolution for Simple Large Objects

If the replicate is defined with an SPL conflict-resolution rule, the SPL routine must return the desired action for each BYTE or TEXT column. When the routine is invoked, information about each BYTE or TEXT column is passed to the routine as five separate fields. The following table describes the fields.

Argument	Description
Column size (INTEGER)	The size of the column (if data exists for this column). NULL if the column is NULL.
BLOB flag [CHAR(1)]	For the local row, the field is always NULL. For the replicated row: ■ D indicates BYTE or TEXT data is sent from the source database server. ■ U indicates BYTE or TEXT data is unchanged on the source database server.
Column type [CHAR(1)]	■ P indicates tablespace data. ■ B indicates blobspace data.
ID of last update server [CHAR(18)]	The ID of the database server that last updated this column for tablespace data For blobspace data: NULL
Last update time (DATETIME YEAR TO SECOND)	For tablespace data: The date and time when the data was last updated. For blobspace data: NULL

For information on creating stored procedures, see the *Informix Guide to SQL: Tutorial*.

If the routine returns an action code of A, D, I, or U, the routine parses the return values of the replicated columns. Each BYTE or TEXT column can return a two-character field. For information about the action codes, refer to "SPL Conflict-Resolution Rule" on page 3-15.

The first character defines the desired option for the BYTE or TEXT column, as the following table shows.

Value	Function
C	Performs a time-stamp check for this column as used by the time-stamp rule
N	Sets the replicate column to NULL
R	Accepts the replicated data as it is received
L	Retains the local data

The second character defines the desired option for blobspace data if the data is found to be undeliverable, as the following table shows.

Value	Function
N	Sets the replicated column to NULL
L	Retains the local data (default)
O	Aborts the row
X	Aborts the transaction

SPL Conflict Resolution for Smart Large Objects

Enterprise Replication handles conflict resolution for smart large objects using the SPL conflict-resolution rule in the same way as for simple large objects. See "Conflict Resolution for Simple and Smart Large Objects" on page 2-21.

Distributing BYTE and TEXT Data

If Enterprise Replication processes a row and discovers undeliverable BYTE or TEXT columns, the following actions can occur:

- Any undeliverable columns are set to NULL if the replication operation is an INSERT and the row does not already exist at the target.

- The old value of the local row is retained if the replication operation is an UPDATE or if the row already exists on the target.

Considerations for Replicating Smart Large Objects

The following conditions apply to replicating smart large objects:

- Enterprise Replication does not support replication of smart large object updates performed outside of a row update.

- After you update a smart large object that is referenced explicitly in the table schema, you must update the referencing row before Enterprise Replication can replicate the updated smart large object. For example:

        UPDATE *table_name* SET *smart_large_object_column* = x

  For more information, see the *Informix Guide to SQL: Syntax.*

- Enterprise Replication replicates updates to in-place smart large objects by sending a new copy of the entire smart large object. Enterprise Replication does not send only the logged changes to update smart large objects.

- Enterprise Replication does not support sharing out-of-row data (multiple references to a smart large object) during replication. If you try to replicate multiple references to the same smart large object on the source database server, Enterprise Replication does not re-create those references on the target database server. Instead, Enterprise Replication creates multiple smart large objects on the target database server.

Replicating Opaque User-Defined Data Types

Enterprise Replication supports built-in Informix data types and opaque user-defined types (UDTs). For a list of supported extensible data types, see "Extensibility Enhancements" on page 6 of the Introduction.

UDT Support Functions

If you plan to replicate opaque user-defined types (UDTs), the UDT designer must provide two support functions, **streamwrite( )** and **streamread( )**.

The purpose of these functions is similar to the existing **send( )** and **receive( )** functions provided for client/server transmissions.

For information on writing these support functions, see the section on Enterprise Replication stream support functions in the *DataBlade API Programmer's Manual*.

When preparing a row that includes any UDT columns to queue to the target system, Enterprise Replication calls the **streamwrite( )** function on each UDT column. The function converts the UDT column data from the in-server representation to a representation that can be shipped over the network. This allows Enterprise Replication to replicate the column without understanding the internal representation of the UDT.

On the target server, Enterprise Replication calls the **streamread( )** function for each UDT column that it transmitted using the **streamwrite( )** function.

Considerations for Replicating Opaque Data Types

The following conditions apply to replicating opaque data types:

- The WHERE clause of the SELECT statement of the participant modifier can reference an opaque UDT as long as the UDT is always stored in row.

- Any UDRs in a WHERE clause can use only parameters whose values can be extracted fully from the logged row images, plus any optional constants.

- All of the columns in the SELECT statement of each participant definition must be actual columns in that table. Enterprise Replication does not support virtual columns (results of UDRs on table columns).

  See "Participant Modifier" on page A-90 for information on the WHERE clause in participant definitions.

- You cannot use SPL routines for conflict resolution if the replicate includes any UDTs in the SELECT statement or if the replicate is defined to replicate only changed columns.

  See "Conflict Resolution" on page 3-11 and "Replicating Only Changed Columns" on page 5-11.

- Enterprise Replication allows you to define replicates on tables that contain one or more UDT columns as the primary key.

  For more information, see the section on primary key constraints in the *Informix Guide to SQL: Syntax*.

Replicating Table Hierarchies

To replicate tables that form a hierarchy, you must define a separate replicate for each table. If you define a replicate on a super table, Enterprise Replication does not automatically create implicit replicate definitions on the subordinate tables.

Tip: Enterprise Replication does not require that the table hierarchies be identical on the source and target servers.

You must use conflict resolution uniformly for all tables in the hierarchy. In other words, either no conflict resolution for all tables or conflict resolution for all tables.

Setting Up and Managing Enterprise Replication

Section II

Selecting the Enterprise Replication System and Network Topology

In This Chapter

This chapter describes types of replication systems provided by Enterprise Replication and discusses the trade-offs associated with performance and data availability.

Selecting the Enterprise Replication System

Enterprise Replication supports the following types of replication systems:

- Primary-Target Replication System
- Update-Anywhere Replication System

Primary-Target Replication System

In the primary-target replication system, the flow of information is in one direction. That is, the primary always sends data to the target. The target does not send data to the primary.

In primary-target replication, all database changes originate at the primary database and are replicated to the target databases. Changes at the target databases are not replicated to the primary.

A primary-target replication system can provide one-to-many or many-to-one replication:

- One-to-many replication

  In the one-to-many (*distribution*) replication, all changes to a primary database server are replicated to many target database servers. Use this replication model when information gathered at a central site must be disseminated to many scattered sites.

- Many-to-one replication

  In many-to-one (*consolidation*) replication, many primary servers send information to a single target server. Use this replication model when many sites are gathering information (for example, local field studies for an environmental study) that needs to be centralized for final processing.
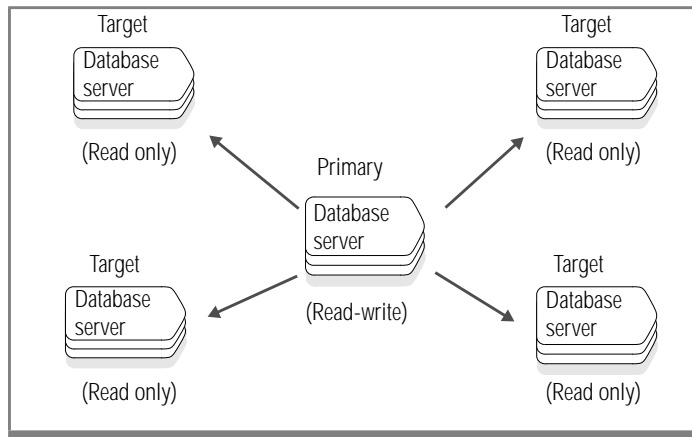
Primary-Target Business Models

Primary-target Enterprise Replication systems support the following business models:

- Data Dissemination
- Data Consolidation
- Workload Partitioning
- Workflow Replication

Data Dissemination

Data dissemination supports business needs where data is updated in a central location and then replicated to read-only sites. This method of distribution can be particularly useful for online transaction processing (OLTP) systems where data is required at several sites, but because of the large amounts of data, read-write capabilities at all sites would cripple the performance of the application. Figure 3-1 illustrates data dissemination.

Figure 3-1
Data Dissemination in a Primary-Target Replication System

Data Consolidation

As businesses reorganize to become more competitive, many choose to consolidate data into one central database server. Data consolidation allows the migration of data from several database servers to one central database server. In Figure 3-2, the remote locations have read-write capabilities while the central database server is read only.
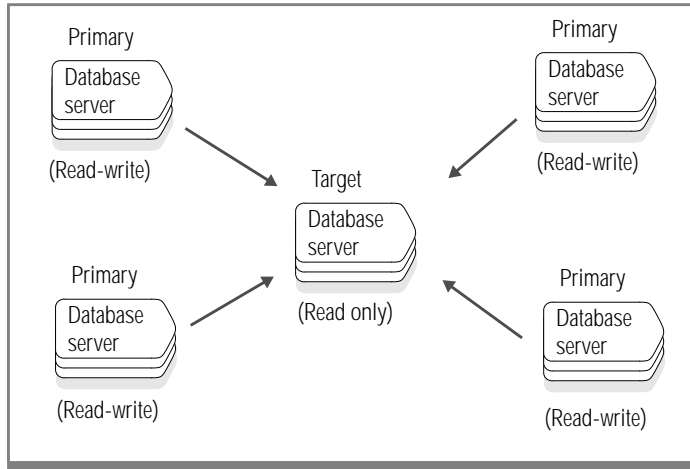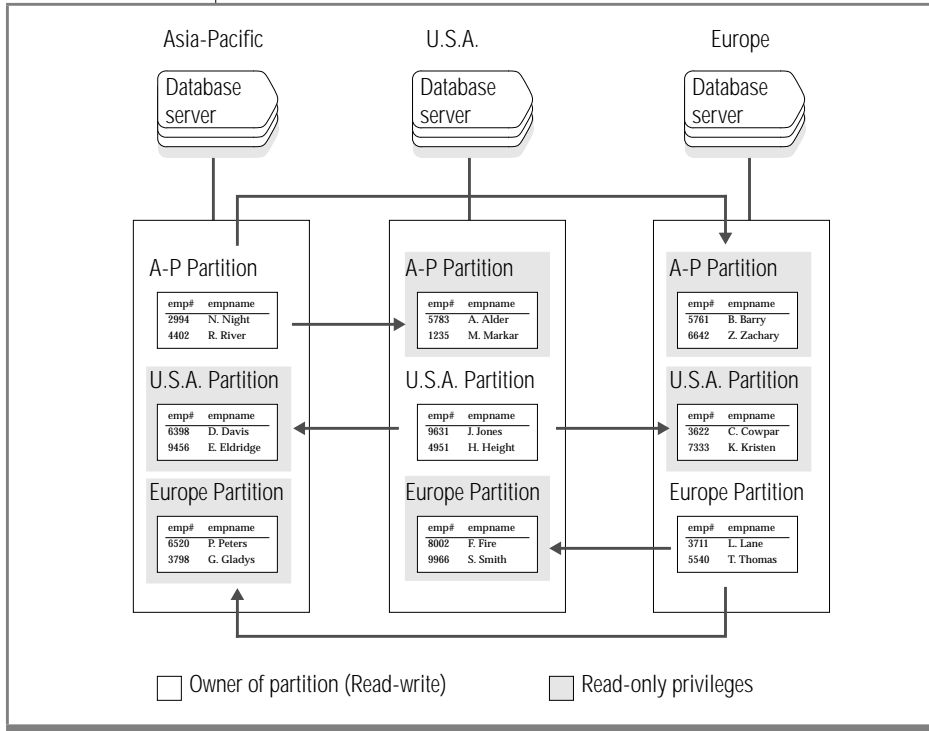

Figure 3-2
Data Consolidation in a Primary-Target Replication System

Businesses can also use data consolidation to off-load OLTP data for decision support (DSS) analysis. For example, data from several OLTP systems can be replicated to a DSS system for read-only analysis. Pay close attention to the configuration of the tables from which data is replicated to ensure that each primary key is unique among the multiple primary database servers.

Workload Partitioning

Workload partitioning gives businesses the flexibility of assigning data ownership at the table-partition level, rather than within an application. Figure 3-3 illustrates workload partitioning.
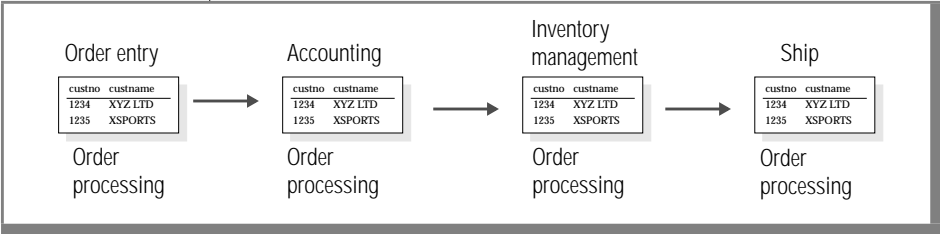
In Figure 3-3, the replication model matches the partition model for the **employee** tables. The Asia-Pacific computer owns the partition and can therefore update, insert, and delete employee records for personnel in its region. The changes are then propagated to the U.S. and European regions. Asia-Pacific can query or read the other partitions locally, but cannot update those partitions locally. This strategy applies to other regions as well.

Workflow Replication

Unlike the data dissemination model, in a workflow-replication system, the data moves from site to site. Each site processes or approves the data before sending it on to the next site.

Figure 3-4 illustrates an order-processing system. Order processing typically follows a well-ordered series of steps: orders are entered, approved by accounting, inventory is reconciled, and the order is finally shipped.

In a workflow-replication system, application modules can be distributed across multiple sites and databases. Data can also be replicated to sites that need read-only access to the data (for example, if order-entry sites want to monitor the progress of an order).

A workflow-replication system, like the primary-target replication system, allows only unidirectional updates. Many facts that you need to consider for a primary-target replication system should also be considered for the workflow-replication system.

However, unlike the primary-target replication system, availability can become an issue if a database server goes down. The database servers in the workflow-replication system rely on the data updated at a previous site. Consider this fact when you select a workflow-replication system.

Primary-Target Considerations

The following sections describe some of the factors to consider when you select a primary-target replication system:

- Administration

    Primary-target replication systems are the easiest to administer because all updates are unidirectional and therefore, no data update conflicts occur. Primary-target replication systems use the *ignore* conflict-resolution rule. See "Conflict-Resolution Rule" on page 3-12.

- Capacity Planning

    All replication systems require you to plan for capacity changes. For more information, see "Preparing Data for Replication" on page 4-18.

- High Availability Planning

    In the primary-target replication system, if a target database server or network connection goes down, Enterprise Replication continues to log information for the database server until it becomes available again. If a database server is unavailable for some time, you might want to remove the database server from the replication system. If the unavailable database server is the read-write database server, you must plan a course of action to change read-write capabilities on another database server. If you require a fail-safe replication system, you should select an update-anywhere replication system.

Update-Anywhere Replication System

In update-anywhere replication, changes made on any participating database server are replicated to all other participating database servers. This capability allows users to function autonomously even when other systems or networks in the replication system are not available.

Figure 3-5 illustrates an update-anywhere replication system.
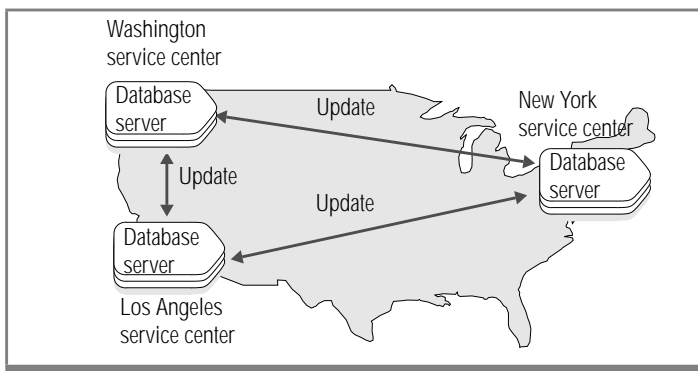
Figure 3-5
Update-Anywhere Replication System

Because each service center can update a copy of the data, conflicts can occur when the data is replicated to the other sites. To resolve update conflicts, Enterprise Replication uses *conflict resolution*. For more information, see "Conflict Resolution" on page 3-11.

Update-Anywhere Considerations

Review the following information before you select your update-anywhere replication system:

- Administration

  Update-anywhere replication systems allow peer-to-peer updates, and therefore require *conflict-resolution* (see "Conflict Resolution" on page 3-11). Update-anywhere replication systems require more administration than primary-target replication systems.

- Information Consistency

  Some risk is associated with delivering consistent information in an update-anywhere replication system. You determine the amount of risk based on the type of conflict-resolution rules and routines you choose with which to resolve conflicts. You can configure an update-anywhere replication system where no data is ever lost; however, you might find that other factors (for example, performance) out-weigh your need for a fail-safe mechanism to deliver consistent information.

- Capacity Planning

  All replication systems require you to plan for capacity changes. For more information, see "Preparing Data for Replication" on page 4-18. In particular, if you choose the time stamp or time stamp plus SPL routine conflict-resolution rule, see "Delete Table Disk Space" and "Shadow Column Disk Space" on page 4-11.

Conflict Resolution

When multiple database servers try to update the same row simultaneously (the time stamp for both updates is the same GMT time), a collision exists ("Apply Replicated Data" on page 1-17). Enterprise Replication must determine which new data to replicate. To solve conflict resolution, you must specify the following for each replicate:

- Conflict-Resolution Rule
- Scope

Conflict-Resolution Rule

Enterprise Replication supports the following conflict-resolution rules.

Conflict Resolution Rule	Effect	Reference
Ignore	Enterprise Replication does not attempt to resolve conflicts.	page 3-13
Time stamp	The row or transaction with the most recent time stamp is applied.	page 3-14
SPL routine	Enterprise Replication uses a routine written in SPL (Stored Procedure Language) that you provide to determine which data should be applied.	page 3-15
Time stamp with SPL routine	If the time stamps are identical, Enterprise Replication invokes an SPL routine that you provide to resolve the conflict.	page 3-14, page 3-15

For all conflict-resolution rules except *ignore*, you must create shadow columns in the tables on both the source and target servers involved in replication. See "Shadow Columns" on page 2-11.

Enterprise Replication supports up to two conflict-resolution rules for each replicate: a primary rule and a secondary rule (if desired). Figure 3-6 shows the valid combinations of Enterprise Replication conflict-resolution rules.

Figure 3-6
Valid Conflict-Resolution Rule Combinations

Primary Rule	Secondary Rule
Ignore	None
Time stamp	None
Time stamp	SPL routine
SPL routine	None

Ignore Conflict-Resolution Rule

The *ignore* conflict-resolution rule does not attempt to detect or resolve conflicts. A row or transaction either applies successfully or it fails. A row might fail to replicate because of standard database reasons, such as a *deadlock* situation, when an application locks rows.

The *ignore* conflict-resolution rule can only be used as a primary conflict-resolution rule and can have either a transaction or a row scope (as described in "Scope" on page 3-18). Figure 3-7 describes the *ignore* conflict-resolution rule.

Figure 3-7
Ignore Conflict-Resolution Rule

Row Exists in Target?	Database Operation		
	Insert	Update	Delete
No	Apply row	Discard row	Discard row
Yes	Discard row	Apply row	Apply row

When a replication message fails to apply on a target, you can spool the information to one or both of the following directories:

- Aborted-transaction spooling (ATS)

  If selected, all buffers in a failed replication message that compose a transaction are written to this directory.

- Row-information spooling (RIS)

  If selected, the replication message for a row that could not be applied to a target is written to this directory.

For more information, see Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."

Time-Stamp Conflict-Resolution Rule

The time-stamp rule evaluates the latest time stamp of the replication against the target and determines a winner.

Tip: All time stamps and internal computations are performed in Greenwich Mean Time (GMT).

The time-stamp resolution rule behaves differently depending on which scope is in effect:

- Row scope

   Enterprise Replication evaluates one row at a time. The row with the most recent time stamp wins the conflict and is applied to the target database servers. If an SPL routine is defined as a secondary conflict-resolution rule, the routine resolves the conflict when the row times are equal.

- Transaction scope

   Enterprise Replication evaluates the most recent row-update time among all the rows in the replicated transaction. This time is compared to the time stamp of the appropriate target row. If the time stamp of the replicated row is more recent than the target, the entire replicated transaction is applied. If a routine is defined as a secondary conflict-resolution rule, it is used to resolve the conflict when the time stamps are equal.

For more information, see "Scope" on page 3-18.

Tip: A secondary routine is invoked only if Enterprise Replication evaluates rows and discovers equal time stamps.

If either type of scope (row or transaction) evaluates rows and discovers equal time stamps, and no routine is defined as a secondary conflict-resolution rule, the replicated row or transaction is applied.

Figure 3-8 on page 3-15 shows how a conflict is resolved based on the latest time stamp with row scope. The time stamp T_{last_update} (the time of the last update) represents the row on the target database server with the last (most recent) update. The time stamp T_{repl} (the time when replication occurs) represents the time stamp on the incoming row.

Enterprise Replication first checks to see if a row with the same primary key exists in either the target table or its corresponding delete table.

Important: *Do not remove the delete tables created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.*

If the row exists, Enterprise Replication uses the latest time stamp to resolve the conflict.

Figure 3-8
Conflict Resolution Based on the Time Stamp

Row Exists on Target?	Time Stamp	Database Operation		
		Insert	Update	Delete
No	n/a	Apply row	Apply row (Convert UPDATE to INSERT)	Apply row (INSERT into Enterprise Replication delete table)
Yes	$T_{last\_update} < T_{repl}$	Apply row (Convert INSERT to UPDATE)	Apply row	Apply row
	$T_{last\_update} > T_{repl}$	Discard row		
	$T_{last\_update} = T_{repl}$	Apply row if no routine is defined as a secondary conflict-resolution rule. Otherwise, invoke the routine.		

The time-stamp conflict-resolution rule assumes time synchronization between cooperating Enterprise Replication servers. For more information, see "Time Synchronization" on page 2-17.

SPL Conflict-Resolution Rule

Tip: *The SPL rule allows you complete flexibility to determine which row prevails in the database. However, for most users, the time-stamp conflict-resolution rule provides sufficient conflict resolution.*

You can assign an SPL routine as the primary conflict-resolution rule. If you use an SPL routine as a secondary conflict-resolution rule, the time-stamp conflict-resolution rule must be the primary rule.

Important: *The owner of an SPL routine used for conflict resolution must be the same as the owner of the table.*

Routines for conflict-resolution must be in SPL. Enterprise Replication does not allow user-defined routines in C or in Java.

Important: *You cannot use SPL routine or time stamp with SPL routine if the replicate is defined to replicate only changed columns or the replicated table contains any extensible data types. See "Replicating Only Changed Columns" on page 5-11 and "Extensibility Enhancements" on page 6 of the Introduction.*

Enterprise Replication passes the following information to an SPL routine as arguments.

Argument	Description
Server name [CHAR(18)]	From the local target row
	NULL if local target row does not exist
Time stamp (DATETIME YEAR TO SECOND)	From the local target row
	NULL if local target row does not exist
Local delete-table indicator [CHAR(1)]	■ Y indicates the origin of the row is the delete table
or	■ K indicates the origin of the row is the replicate-key delete row (for a key update that is sent as a key delete and a key insert) because the local target row with the old key does not exist
Local key delete-row indicator [CHAR(1)]	
Server name [CHAR(18)]	Of the replicate source
Time stamp (DATETIME YEAR TO SECOND)	From the replicated row
Replicate action type [CHAR(1)]	I- insert
	D- delete
	U- update
Local row data returned in regular SQL format	Where the regular SQL format is taken from the SELECT clause of the participant list
Replicate row data after-image returned in regular SQL format	Where the regular SQL format is taken from the SELECT clause of the participant list

The routine must set the following arguments before the routine can be applied to the replication message.

Argument	Description
An indicator of the desired database operation to be performed [CHAR(1)]	Same as the replicate action codes with the following additional codes
	■ A - Accept the replicated row and apply the column values returned by the SPL routine.
	For example, if Enterprise Replication receives an insert and the row already exists locally, the insert is converted to an update
	■ S - Accept the replicated row and apply the column values as received from the other site.
	For example, if Enterprise Replication receives an insert and the row already exists locally, the insert fails at the time Enterprise Replication tries to apply the transaction to the database, and the transaction aborts with a SQL error.
	■ O - Discard the replicated row.
	■ X - Abort the transaction.
A non-zero integer value to request logging of the resolution and the integer value in the spooling files (INTEGER)	Logging value takes effect only if logging is configured for this replicate.
The columns of the row to be applied to the target table replicate action type in regular SQL format	This list of column values is not parsed if the replicate action type that the routine returns is S, O, or X.

You can use the arguments to develop application-specific routines. For example, you can create a routine in which a database server always wins a conflict regardless of the time stamp.

The following list includes some items to consider when you use an SPL routine for conflict resolution:

■ Any action that a routine takes as a *result* of replication does not replicate.

■ You cannot use an SPL routine to start another transaction.

■ Frequent use of routines might affect performance.

In addition, you must determine when the SPL routine executes:

■ An *optimized* SPL routine is called only when a collision is detected and the row to be replicated fails to meet one of the following two conditions:

❑ It is from the same database server that last updated the local row on the target table.

❑ It has a time stamp greater than or equal to that of the local row.

■ A *nonoptimized* SPL routine executes every time Enterprise Replication detects a collision. By default, SPL routines are nonoptimized.

For information on specifying that the SPL routine is optimized, see "Conflict Options" on page A-11.

Tip: Do not assign a routine that is not optimized as a primary conflict-resolution rule for applications that usually insert rows successfully.

Scope

Each conflict-resolution rule behaves differently depending on the *scope*. Enterprise Replication uses the following scopes:

■ Row scope

When you choose a row scope, Enterprise Replication evaluates one row at a time. It only applies replicated rows that win the conflict resolution with the target row. If an entire replicated transaction receives row-by-row evaluation, some replicated rows are applied while other replicated rows might not be applied.

■ Transaction scope

When you choose a transaction scope, Enterprise Replication applies the entire transaction if the replicated transaction wins the conflict resolution. If the target wins the conflict (or other database errors are present), the entire replicated transaction is not applied.

A transaction scope for conflict resolution guarantees transactional integrity.

Important: *Enterprise Replication defers some constraint checking on the target tables until the transaction commits. If a unique constraint or foreign-key constraint violation is found on any row of the transaction at commit time, the entire transaction is rejected (regardless of the scope) and, if you have ATS set up, written to the ATS directory. For more information, see "Aborted Transaction Spooling Files" on page 8-3.*

Choosing a Replication Network Topology

Enterprise Replication *topology* describes connections that replication servers make to interact with each other. You must define the replication topology within the topology of the physical network, but the replication topology is not synonymous with the physical network topology. The order in which you define the database servers for replication and the options that you use when you define the database server determine the replication topology.

Enterprise Replication supports two types of network topology:

- Fully Connected Topology
- Hierarchical Replication Topologies

The topology that you choose influences the types of replication that you can use. The next sections describe the topologies that Enterprise Replication supports.

Fully Connected Topology

Fully connected replication topology indicates that all database servers connect to each other and that Enterprise Replication establishes and manages the connections. Replication messages are sent directly from one database server to another. No additional routing is necessary to deliver replication messages. Figure 3-9 shows a fully connected replication topology. Each database server connects directly to every other database server in the replication environment.
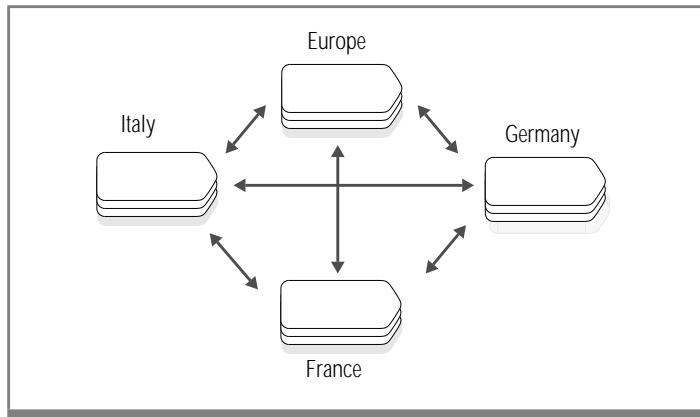
Figure 3-9
Fully Connected Topology

Hierarchical Replication Topologies

Enterprise Replication provides two types of Hierarchical Routing (HR) topology:

- Hierarchical Tree
- Forest of Trees

HR Topology Terminology

Enterprise Replication uses the terms in the Figure 3-10 to describe HR topology.

Figure 3-10
Replication Topology Terms

Term	Definition
Root server	An Enterprise Replication server that is the uppermost level in a hierarchically organized set of information. The root is the point from which database servers branch into a logical sequence. All root database servers within Enterprise Replication must be fully interconnected.
Nonroot server	An Enterprise Replication server that is not a root database server but that has a complete global catalog and is connected to its parent and to its children.
Tree	A data structure that contains database servers that are linked in a hierarchical manner. The topmost node is called the root. The root can have zero or more *child* database servers; the root is the *parent* database server to its children.
Parent-child	A relationship between database servers in a tree data structure in which the parent is one step closer to the root than the child.
Leaf server	A database server that has a limited catalog and no children.

A *root* server is fully connected to all other root servers. It has information about all other replication servers in its replication environment. Figure 3-9 on page 3-20 shows an environment with four root servers.

A *nonroot server* is similar to a root server except that it forwards all replicated messages for other root servers (and their children) through its parent. All nonroot servers are known to all root and other nonroot servers. A nonroot server might or might not have children. All root and nonroot servers are aware of all other servers in the replication environment.

Important: *In HR topologies, Enterprise Replication specifies the synchronization server as the new server's parent in the current topology. For more information, see "Customizing the Replication Server Definition" on page 5-6 and "cdr define server" on page A-20.*

Hierarchical Tree

A *hierarchical tree* consists of a root database server and one or more database servers organized into a tree topology. The tree contains only one root, which has no parent. Each database server within the tree references its parent. A database server that is not a parent is a leaf. Figure 3-11 illustrates a replication tree.
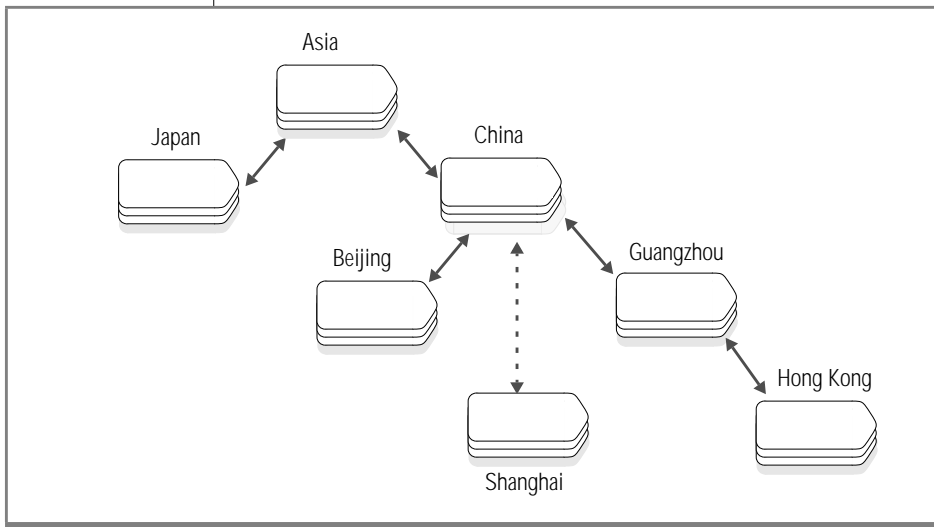
Figure 3-11
Hierarchical Tree Topology

In Figure 3-11, the parent-child relationship within the tree is as follows:

- **Asia** is the parent of **China** and **Japan**.
- **China** is the child of **Asia** *and* the parent of **Beijing**, **Shanghai**, and **Guangzhou**.
- **Guangzhou** is the child of **China** *and* the parent of **Hong Kong**.

Asia is the root database server. **Japan**, **China**, and **Guangzhou** are nonroot database servers. You can define **Beijing**, **Shanghai**, and **Hong Kong** as either nonroot database servers or leaf database servers, depending on how you plan to use them. The dashed connection from **China** to **Shanghai** indicates that Shanghai is a leaf server.

Forest of Trees

A *forest of trees* consists of several hierarchical trees whose root database servers are fully connected. Each hierarchical tree starts with a root database server. The root database servers transfer replication messages to the other root servers for delivery to its child database servers. Figure 3-12 shows a forest of trees.
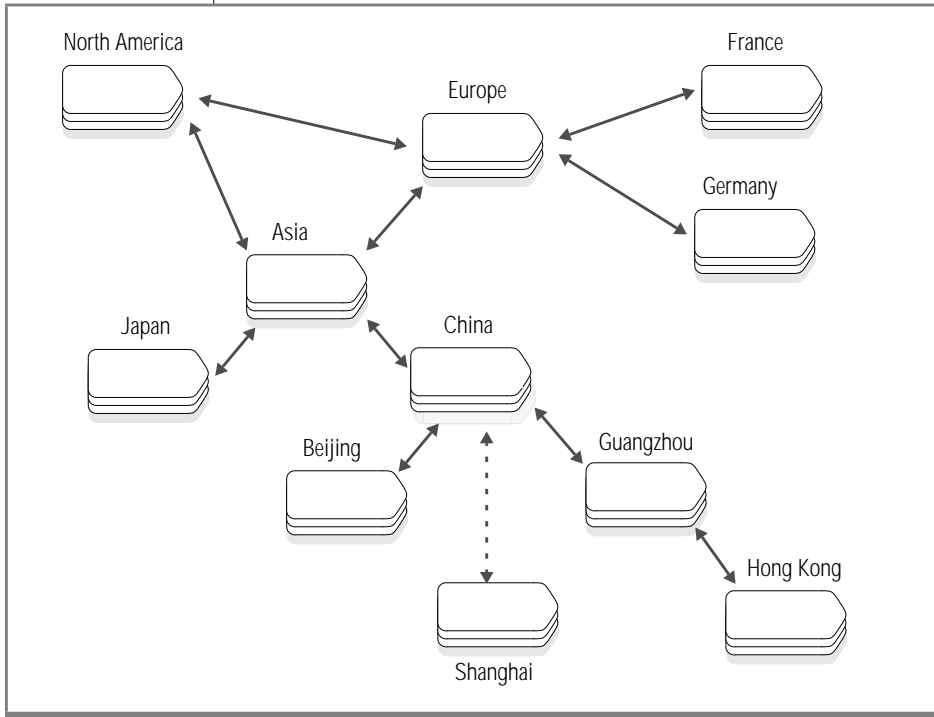
Figure 3-12
Forest-of-Trees Topology

In Figure 3-12, **North America**, **Asia**, and **Europe** are root database servers. That is, they are fully connected with each other. **France** and **Germany** are in a tree whose root is **Europe**. **Asia** is the root for the six database servers in its tree.

In a forest of trees, all replication messages from one tree to another must pass through their roots. For example, a replication message from **Beijing** to **France** must pass through **China**, **Asia**, and **Europe**.

Organizing the database servers in a hierarchical tree or a forest of trees greatly reduces the number of physical connections that are required to make a replication system. If all the database servers in Figure 3-12 were fully connected, instead of being organized in trees, 55 connections would be required.

Preparing the Replication Environment

In This Chapter

This chapter covers the steps to take to prepare your environment for replicating data with Enterprise Replication, which involves preparing the network environment, the disk, the server environment, and the data.

Preparing the Network Environment

For more information on preparing the network environment, see the chapter on client/server connectivity in the *Administrator's Guide for Informix Dynamic Server*. See Appendix E, "Replication Examples," for a sample setup.

To prepare your network environment

1. Set up the hosts file.

    For information, see "Setting Up the Hosts File" on page 4-4.

2. Set up the services file.

    For information, see "Setting Up the Services File" on page 4-4.

3. Set up the trusted environment

    For information, see "Setting Up the Trusted Environment" on page 4-5.

4. Verify the SQLHOSTS information.

    For information, see "Verifying SQLHOSTS" on page 4-5.

5. Test the network environment.

    For information, see "Testing the Network Environment" on page 4-8.

Setting Up the Hosts File

First, make sure the **hosts** file includes the IP addresses and system names for all database servers involved in Enterprise Replication.

Important: *If you are using Domain Name Service (DNS) to identify IP addresses and system names, you do not need to configure the **hosts** file.*

The **hosts** file is in the following location.

Operating System	File
UNIX	**/etc/hosts**
Windows	**%WINDIR%\system32\drivers\etc\hosts**

Important: *Leave a blank line at the end of the **hosts** file on Windows.*

For example, your **hosts** file might look like the following:

```
123.456.789.1 sydney
123.456.789.2 melbourne
```

Setting Up the Services File

Next, make sure that the **services** file includes the port numbers and service names for all the database servers involved in Enterprise Replication. The **services** file is in the following location.

Operating System	File
UNIX	**/etc/services**
Windows	**%WINDIR%\system32\drivers\etc\services**

Important: *Leave a blank line at the end of the **services** file on Windows.*

For example, your **services** file might look like the following:

```
sydney 5327/tcp
melbourne 5327/tcp
```

If the database servers reside on the same system, you must provide unique port numbers for each.

Setting Up the Trusted Environment

To establish the trust relationship for all users, set up the **hosts.equiv** file. The **hosts.equiv** file is in the following location.

Operating System	File
UNIX	**/etc/hosts.equiv**
Windows	**%WINDIR%\system32\drivers\etc\hosts.equiv**

For example, your **hosts.equiv** file might look like the following:

```
sydney
melbourne
```

Tip: Instead of allowing access to all users, you can set up .rhosts files in the home directory of specific users. See your operating system documentation for more information.

Verifying SQLHOSTS

Make sure that the SQLHOSTS file is set up properly on each server participating in replication.

Setting up Database Server Groups

Enterprise Replication requires that all database servers participating in replication be members of database server groups. Each server in the enterprise must have a unique identifier. The database server group uniquely identifies the server.

Typically, a server group includes only one database server. However, if the computer has multiple network protocols or network interface cards, the server group includes all aliases for the database server. Enterprise Replication treats the server group as one object, whether it includes one or several database server names.

All Enterprise Replication commands and options (except the --**connect** option which can use both) use the name of the *database server group* instead of the more familiar *database server* name (that is, the name specified by the **INFORMIXSERVER** environment variable) for all references to database servers. This manual also refers to a database server group as a *server group*.

This manual uses the convention that the name of a database server group is **g_** followed by the name of a database server that is in the group. This use of **g_** is only a convention; **g_** is not required syntax.

UNIX

Database Server Groups on UNIX

On UNIX, a database server group is defined in the **sqlhosts** file. The following example shows a very simple **sqlhosts** file for four Enterprise Replication servers, **john**, **paul**, **george**, and **ringo** and their database server groups. The first line describes the database server group **g_john**, which includes the database server **john**, and so on.

dbservername	nettype	hostname	servicename	options
g_john	group	–	–	i=143
john	ontlitcp	sydney.australia.com	10110	g=g_john
g_paul	group	–	–	i=144
paul	ontlitcp	melbourne.australia.com	2939	g=g_paul
g_george	group	–	–	i=145
george	ontlitcp	perth.australia.com	5329	g=g_george
g_ringo	group	–	–	i=146
ringo	ontlitcp	brisbane.australia.com	10101	g=g_ringo

The following table describes the fields in the **sqlhosts** example above.

dbservername	Database server group name or database server name
nettype	Type of connection (composed of the database server product, interface type, and network protocol)

(1 of 2)

hostname	The name of the computer where the database server resides
servicename	The service name or port number entry in the **services** file
options	■ The **g** option specifies the name of the group to which the database server belongs.
	■ The **i** option specifies a unique identifier for the database server. Make sure that this identifier is consistent for the database server across all nodes in the enterprise.

(2 of 2)

Important: The network connection entry should appear immediately after the database server group definition.

Important: If you use both DBSERVERNAME and DBSERVERALIASES, DBSERV-ERNAME should refer to the network connection and not to a shared-memory connection. For information about database server aliases, refer to the Adminis-trator's Guide for Informix Dynamic Server.

For more information about database server groups and setting up SQLHOSTS, see the chapter on client/server communications in the *Adminis-trator's Guide for Informix Dynamic Server*.

WIN NT/2000

Database Server Groups on Windows

For information about preparing the SQLHOSTS connectivity information on Windows, see Appendix F, "SQLHOSTS Registry Key."

*Important: Informix strongly recommends that you use Informix Server Adminis-trator (ISA), rather than **regedt32**, to set up the SQLHOSTS registry key and database server group registry key on your Windows system. In addition, ISA allows you to administer your replication system from a web browser. For more information, see the Documentation Notes described in "Additional Documentation" on page 15 of the Introduction.*

Hierarchical Routing Topologies and SQLHOSTS

For hierarchical routing (HR) topologies:

- Root and nonroot servers must each have complete SQLHOSTS server group information for the entire enterprise.
- The leaf servers require only the SQLHOSTS connectivity information for itself and its parent (hub).

Root and nonroot servers contain the complete global catalog; leaf servers do not. For more information, see "HR Topology Terminology" on page 3-21 and "Global Catalog" on page 2-6.

Testing the Network Environment

Once you have verified the network setup information, test the network environment.

To test the network environment

1. Verify the network connection.

   Use the **ping** command to test the connection between two systems. For example, from **sydney**, test the connection to **melbourne**:

   ```
   ping melbourne
   ```

2. Test the trusted environment.

   a. Run **dbaccess**.

   b. Select the **Connection** menu option.

   c. Select the **Connect** menu option.

   d. Connect to the server group name and the server name of the other hosts.

      For example, if you are running **dbaccess** on **sydney**, and you are testing the connection to a database server on **melbourne**, select **paul** and **g_paul**.

   e. When prompted for the USER NAME, press ENTER.

   If you can connect to the host database server, the host server is trusted for user **informix**.

   For more information, see the *DB-Access User's Manual*.

Preparing the Disk

Preparing your disk for Enterprise Replication includes the following:

- Planning for Disk Space Requirements
- Setting Up Send and Receive Queue Spool Areas
- Creating ATS and RIS Directories

Planning for Disk Space Requirements

Enterprise Replication requires additional disk space for storing the logical logs and, depending on your conflict-resolution configuration, delete tables and shadow columns.

Logical Log Configuration Disk Space

The database server uses the logical log to store a record of changes to the data since the last archive. Enterprise Replication requires the logical log to contain entire row images for updated rows, including deleted rows.

The database server normally logs only columns that have changed. This behavior is called the logical-log record reduction option. Enterprise Replication deactivates this option for tables that participate in replication. (The logical-log record reduction option remains enabled for tables that do *not* participate in Enterprise Replication.) Enterprise Replication logs all columns, not only the columns that have changed, which increases the size of your logical log.

To determine the size of your logical log, examine your data activity for normal operations and for the replication system you defined. Keep in mind that defining replication on a table causes Enterprise Replication to deactivate log reduction for that table, and that your transactions might log more data.

Important: *Enterprise Replication performs internal cleanup tasks based on how often the log files switch. If the log files switch too frequently, Enterprise Replication might perform excessive cleanup work.*

Logical Log Configuration Guidelines

Use the following guidelines when configuring your logical log files:

- Make sure that all logical log files are approximately the same size.
- Make the size of the logical log files large enough so that the database server switches log files no more than once every 15 minutes during normal processing.
- Plan to have sufficient logical-log space to hold at least four times the maximum transaction size.
- Set LTXEHWM (long-transaction, exclusive-access, high-water mark) 30 percent larger than LTXHWM (long-transaction high-water mark).

Important: *If you specify that the database server allocate logical log files dynamically (DYNAMIC_LOGS), Informix recommends that you set LTXEHWM to no higher than 70 when using Enterprise Replication.*

For more information about logical logs and these configuration parameters, see *Informix Administrator's Reference* and *Administrator's Guide for Informix Dynamic Server*.

Delete Table Disk Space

If you use the time stamp or time stamp and SPL routine conflict-resolution rules, Enterprise Replication creates *delete tables* to keep track of modified rows for conflict resolution. (Enterprise Replication creates delete tables only for tables that have replicates defined with a conflict-resolution rule other than *ignore*.) Delete tables handle conflicts such as when a DELETE or UPDATE finds no corresponding row on the target.

For more information, see "Conflict-Resolution Rule" on page 3-12.

Delete tables are created on the database server where the data originates and on all the database servers to which data gets replicated. Delete tables are stored in the same dbspaces, using the same fragmentation strategy, as their base tables.

To determine the disk space requirements to accommodate delete tables, estimate how many rows will be deleted or modified. For example, if the base table has 100 megabytes of data, but only half the rows might be deleted or modified, then 50 megabytes is a reasonable estimate for the size of the delete table.

Important: *Do not remove the delete tables created by Enterprise Replication. The delete table is automatically removed when the last replicate defined with conflict resolution is deleted.*

Shadow Column Disk Space

When you define a replicate that uses any conflict-resolution rule except *ignore*, you must define *shadow columns* (CRCOLS) with the WITH CRCOLS clause. The shadow columns, **cdrserver** and **cdrtime**, store server and time stamp information that Enterprise Replication uses for conflict resolution. The two shadow columns are integers, which adds a total of 8 bytes to each row in the table involved in a replicate that uses conflict resolution.

Tip: *If you plan to use only the ignore conflict-resolution rule, you do not need to define the **cdrserver** and **cdrtime** shadow columns.*

For more information, see "Conflict-Resolution Rule" on page 3-12 and "Preparing Tables for Conflict Resolution" on page 4-21.

Setting Up Send and Receive Queue Spool Areas

The term *data queue* refers to both the *send queue* and the *receive queue*. Enterprise Replication collects information from the logical logs and places the data to be transferred in the send queue. Then Enterprise Replication transfers the contents of the send queue to the receive queue on the target server. Enterprise Replication on the target reads the data from the receive queue and applies the changes to the tables on the target server.

The send and receive queues reside in memory and are managed by the Reliable Queue Manager (RQM). The CDR_QUEUEMEM configuration parameter ("CDR_QUEUEMEM" on page B-7) specifies the amount of memory space that is available for the data queues.

When a queue in memory fills (for the receive queue, this only occurs with large transactions), the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of *transaction records* (headers that contain internal information for Enterprise Replication) and *row data* (the actual replicated data). Spooled transaction records are stored in the transaction tables in a single dbspace. Spooled row data is stored in a single sbspace.

Important: *To prevent the send and receive queues from spooling to disk, see "Preventing Memory Queues from Overflowing" on page 8-13.*

Transaction Record dbspace

By default, the transaction records are stored in the root dbspace. Because Enterprise Replication and other database servers become unavailable if the root dbspace fills, Informix recommends that you define a single, separate dbspace for the send and receive queue transaction records before you define the replication server.

To determine the size of your transaction record dbspace, you must determine the estimated number of transactions in a given period. You should allocate 110 bytes per transaction to the dbspace and allocate enough disk space to store 24 hours of transaction records. For example, if your network is down for 24 hours, and you estimate that you will log 1000 transactions each day, the size of the transaction record dbspace should be at least 108 kilobytes (110 bytes * 1000 transactions / 1024).

To create the transaction record dbspace, use **onspaces -c**. For example, to create a 110 kilobyte dbspace called **er_dbspace** using raw disk space on UNIX with an offset of 0, enter:

```
onspaces -c -d er_dbspace -p /dev/raw_dev1 -o 0 -s 110
```

The pathname for the dbspace cannot be longer than 256 bytes.

Set the CDR_QHDR_DBSPACE configuration parameter ("CDR_QHDR_DBSPACE" on page B-8) in the ONCONFIG file to the location of the transaction record dbspace (er_dbspace, in this example).

Warning: *Do not change the value of CDR_QHDR_DBSPACE after you initialize Enterprise Replication on a server.*

For information on creating dbspaces, see the chapter on managing disk space in the *Administrator's Guide for Informix Dynamic Server* and the utilities chapter in the *Informix Administrator's Reference*.

Row Data sbspace

Because replicated data might include UDT and CLOB/BLOB data types, the
spooled row data is stored as a smart large object in an sbspace.

*Important: You must create an sbspace for the spooled row data and set the
CDR_QDATA_SBSPACE configuration parameter to the location of this sbspace
before starting Enterprise Replication.*

Informix recommends that you create a separate sbspace for Enterprise
Replication and that the sbspace size be sufficiently large.

*Warning: When the send queue sbspace fills, Enterprise Replication hangs until you
either resolve the problem that is causing Enterprise Replication to spool and/or
allocate additional disk space to the sbspace. For more information, see "Preventing
Memory Queues from Overflowing" on page 8-13.*

To determine the size of your spooled row data sbspace, determine your log
usage and then consider how much data you can collect if your network goes
down. For example, assume that you usually log 40 megabytes of data each
day, but only 10 percent of that is replicated data. If your network is down for
24 hours and you estimate that you will log 4 megabytes of replicated data
each day, the sbspace you identify for the spooled row data must be at least
4 megabytes.

WIN NT/2000

On Windows, increase the resulting size of the sbspace by approximately a
factor of two. (The default page size, the way that data maps onto a page, and
the number of pages written to disk are different on Windows.) ♦

To create the row data sbspace, use **onspaces** -**c**. For example, to create a
4-megabyte sbspace, called **er_sbspace**, using raw disk space on UNIX with
an offset of 0, enter:

```
onspaces -c -S er_sbspace -p /dev/rdsk/c0t1d0s4 -o 0 -s 4000\
    -m /dev/rdsk2/c0t1d0s4 0 \
    -Df "AVG_LO_SIZE=2,LOGGING=OFF"
```

The pathname for the sbspace cannot be longer than 256 bytes.

The -**m** option specifies the location and offset of the sbspace mirror. The -**Df** option specifies default behavior of the smart large objects stored in the sbspace:

- AVG_LO_SIZE (average large object size)

    Set this parameter to the expected average transaction size (in kilobytes). The database server uses this value to calculate the metadata size. The minimum value for AVG_LO_SIZE is 2 kilobytes, which is appropriate for Enterprise Replication in most cases. (The default value of AVG_LO_SIZE is 32 kilobytes.) If you set AVG_LO_SIZE to larger than the expected transaction size, you might run out of metadata space. If you set AVG_LO_SIZE too small, you might waste space on metadata.

- LOGGING

    Informix recommends that you set this parameter to OFF to turn off logging for the smart large object, but that you mirror the sbspace. Because Enterprise Replication writes spooled data to disk before the replay pointer is updated, there is no need to log the row data sbspace. If you set LOGGING to OFF and Enterprise Replication spools to disk, you experience enhanced performance because Enterprise Replication writes less data to disk.

    After creating the sbspace and sbspace mirror, you can use the CDR_SBFLAGS configuration parameter ("CDR_QDATA_SBFLAGS" on page B-10) in the ONCONFIG file to override the current logging setting for the CDR_SBSPACE smart large object.

For information on creating and mirroring sbspaces, see the chapter on managing disk space in the *Administrator's Guide for Informix Dynamic Server* and the utilities chapter in the *Informix Administrator's Reference*.

Set the CDR_QDATA_SBSPACE configuration parameter ("CDR_QDATA_SBSPACE" on page B-9) in the ONCONFIG file to the location of the row data sbspace (er_sbspace, in this example).

Warning: Do not change the value of CDR_QDATA_SBSPACE after you initialize Enterprise Replication.

Creating ATS and RIS Directories

The Aborted Transactions Spooling (ATS) and Row Information Spooling (RIS) files contain information about failed transactions and aborted rows.

If you set up ATS and RIS, Enterprise Replication writes ATS and RIS files to directories on the system:

- ATS files

  If you are using primary-target replication, create the ATS directory on the target system. If you are using update-anywhere replication ("Update-Anywhere Replication System" on page 3-10) and have conflict resolution ("Conflict Resolution" on page 3-11) enabled, create the ATS directory on all participating replication systems.

- RIS files

  If you have conflict resolution enabled, create the RIS directory on all participating replication systems.

The default location for these directories is **/tmp** (UNIX) or **\tmp** (Windows). Informix recommends that you specify a location other than **/tmp** or **\tmp** for the spooling directories.

Create the new location for these directories before you define the server for replication.The pathnames for the ATS and RIS directories can be no longer than 256 characters.

For information about ATS and RIS, refer to Chapter **8**, "Monitoring and Troubleshooting Enterprise Replication."

Preparing the Database Server Environment

To configure the database server environment, verify that the following environment variables are set correctly:

- INFORMIXDIR is set to the full path of the Informix directory.
- INFORMIXSERVER is set to the name of the default database server.

- INFORMIXSQLHOSTS is set to the full path to the SQLHOSTS file (UNIX) or the SQLHOSTS registry host machine (Windows).

For more information, see the *Informix Administrator's Reference*.

In addition, in the ONCONFIG file for each database server, make sure that the following configuration parameters are set:

- DBSERVERNAME is set to the correct database server.

Important: If you use both DBSERVERNAME and DBSERVERALIASES, DBSERVERNAME should refer to the TCP connection and not to a shared-memory connection. For information about database server aliases, refer to the Administrator's Guide for Informix Dynamic Server.

- CDR_QUEUEMEM ("CDR_QUEUEMEM" on page B-7) is set to the maximum amount of memory to be used for the send and receive queues.
- CDR_QHDR_DBSPACE ("CDR_QHDR_DBSPACE" on page B-8) is set to the location of the transaction record dbspace.

- CDR_QDATA_SBSPACE ("CDR_QDATA_SBSPACE" on page B-9) is set to the location of the row data sbspace.

Important: If the CDR_QDATA_SBSPACE configuration parameter is not set in ONCONFIG or the sbspace name specified by CDR_QDATA_SBSPACE is invalid, Enterprise Replication fails to define the server.

■   CDR_SERIAL ("CDR_SERIAL" on page B-11) is set to enable control over generating values for serial columns in tables defined for replication.

For more information, see "SERIAL Data Types and Primary Keys" on page 2-12.

Preparing Data for Replication

The goal of data replication is to provide identical, or at least consistent, data on multiple database servers. This section describes how to prepare the information in your databases for replication.

When you define a new replicate on tables with existing data on different database servers, the data might not be consistent. Similarly, if you add a participant to an existing replicate, you must ensure that all the databases in the replicate have consistent values.

For more information, see "Data Preparation Example" on page 4-24.

Preparing Consistent Data

In most cases, preparing consistent data simply requires that you decide which of your databases has the most accurate data and then copy that data onto the target database. If the target database already has data, for data consistency, you must remove that data before adding the copied data. See "Loading and Unloading Data" on page 4-22 for information on loading the data.

Blocking Replication

You might need to put data into a database that you do not want replicated, perhaps for a new server or because you had to drop and re-create a table.

To block replication while you prepare a table, use the BEGIN WORK WITHOUT REPLICATION statement. This starts a transaction that does not replicate to other database servers.

The following code fragment shows how you might use this statement:

```
BEGIN WORK WITHOUT REPLICATION
LOCK TABLE office
DELETE FROM office WHERE description = 'portlandR_D'
COMMIT WORK
```

The following list indicates actions that occur when a transaction starts with BEGIN WORK WITHOUT REPLICATION:

- SQL does not generate any values for the shadow columns (**cdrserver** and **cdrtime**) for the rows that are inserted or updated within the transaction. You must supply values for these columns with the explicit column list. You must supply these values even if you want the column values to be NULL.

- To modify a table with shadow columns that is already defined in Enterprise Replication, you must explicitly list the columns to be modified. The following two examples show an SQL statement and the correct changes to the statement to modify columns:

  - ❏ If **table_name1** is a table defined for replication, you must change the following statement:

    ```
    LOAD FROM filename INSERT INTO table_name1;
    ```
    to:
    ```
    LOAD FROM filename INSERT INTO table_name1 \
       (list of columns);
    ```

    The list of columns must match the order and the number of fields in the load file.

  - ❏ If **table_name3** and **table_name4** are tables defined for replication with the same schema, you must change the following statement:

    ```
    INSERT INTO table_name3 SELECT * FROM table_name4;
    ```

    to an explicit statement, where *col1*, *col2*,..., *colN* are the columns of the table:

    ```
    INSERT INTO table_name3 VALUES
        (cdrserver, cdrtime, col1, ..., colN)
       cdrserver, cdrtime *
      FROM table_name4;
    ```

    The shadow columns (**cdrserver** and **cdrtime**) are not included in an * list.

For more information about these statements, refer to the *Informix Guide to SQL: Syntax*.

Using DB-Access to Begin Work Without Replication

The following example shows how to use DB-Access to begin work without replication as well as update the Enterprise Replication shadow columns **cdrserver** and **cdrtime**:

```
DATABASE adatabase;
BEGIN WORK WITHOUT REPLICATION
INSERT into mytable (cdrserver, cdrtime, col1, col2, ....)
   VALUES (10, 845484154, value1, value2, ....);
UPDATE mytable
   SET cdrserver = 10, cdrtime = 945484154
   WHERE col1 > col2;
COMMIT WORK
```

Using ESQL/C to Begin Work Without Replication

The following example shows how to use ESQL/C to begin work without replication as well as update the Enterprise Replication shadow columns **cdrserver** and **cdrtime**:

```
MAIN (argc, argv)
   INT   argc;
   CHAR  *argv[];
{
   EXEC SQL CHARstmt[256];
   EXEC SQL database mydatabase;

   sprintf(stmt, "BEGIN WORK WITHOUT REPLICATION");
   EXEC SQL execute immediate :stmt;

   EXEC SQL insert into mytable (cdrserver, cdrtime,
   col1, col2, ...)
   values (10, 845494154, value1, value2, ...);

   EXEC SQL update mytable
   set cdrserver = 10, cdrtime = 845494154
   where col1 > col2;
   EXEC SQL commit work;
}
```


Important: You must use the following syntax when you issue the BEGIN WORK WITHOUT REPLICATION statement from ESQL/C programs. Do not use the '$' syntax.

```
sprintf(stmt, "BEGIN WORK WITHOUT REPLICATION");
EXEC SQL execute immediate :stmt;
```

Preparing to Replicate Opaque User-Defined Types

For Enterprise Replication to be able to replicate opaque user-defined types (UDTs), the UDT designer must provide two support functions, **streamwrite( )** and **streamread( )**. For more information, see "Replicating Opaque User-Defined Data Types" on page 2-25.

Preparing Tables for Conflict Resolution

To use any conflict-resolution rule other than *ignore*, you must define the shadow columns, **cdrserver** and **cdrtime**.

*Tip: If you plan to use only the ignore conflict-resolution rule, you do not need to define the **cdrserver** and **cdrtime** shadow columns.*

For more information about update-anywhere and conflict resolution, see "Update-Anywhere Replication System" on page 3-10.

To define the shadow columns in your table, use the following statements:

- For new tables, use:

    ```
    CREATE TABLE table_name WITH CRCOLS;
    ```

- For existing tables, use:

    ```
    ALTER TABLE table_name ADD CRCOLS;
    ```

Important: If a table already participates in Enterprise Replication, you must stop replication before altering it with the ADD CRCOLS clause. For more information, see "Stopping a Replicate" on page 6-8.

To drop the **cdrserver** and **cdrtime** shadow columns, use:

```
ALTER TABLE table_name DROP CRCOLS;
```

Tip: The ADD CRCOLS and DROP CRCOLS clauses to the ALTER TABLE statement are now processed as in-place alters in most cases. For more information, see "Performance Improvements" on page 7 of the Introduction and the section on in-place alters in the Performance Guide. For more information on CREATE TABLE and ALTER TABLE, see the sections in the Informix Guide to SQL: Syntax.

Preparing Logging Databases

Databases on all server instances involved in replication must be created with logging. Informix recommends that you use unbuffered logging. For more information, see "Unbuffered Logging" on page 2-10.

Loading and Unloading Data

For loading data, you can use the following tools:

- High-Performance Loader
- onunload and onload Utilities
- dbexport and dbimport Utilities
- UNLOAD and LOAD Statements

When you unload and load data, you must use the same type of utility for both the unload and load operations. For example, you cannot unload data with the **onunload** utility and then load the data with a LOAD statement.

If the table that you are preparing for replication is in a database that already uses replication, you might need to block replication while you prepare the table. See "Blocking Replication" on page 4-18 for information on how to do this.

If a table that you plan to replicate includes the shadow columns, **cdrserver** and **cdrtime**, the statements that you use for unloading the data must explicitly name the shadow columns. If you use * FROM *table_name* to the data to unload, the data from the shadow columns will not be unloaded. To include the shadow columns in the unloaded data, use a statement like the following:

```
SELECT cdrserver, cdrtime, * FROM table_name
```

For more information, see "Shadow Columns" on page 2-11.

High-Performance Loader

The High-Performance Loader (HPL) provides a high-speed tool for moving data between databases.

How you use the HPL depends on how you defined the tables to replicate. If the table definition included the WITH CRCOLS clause, you must take special steps when you unload the data.

If the table contains shadow columns, you must:

- Include the **cdrserver** and **cdrtime** columns in your map when you load the data.
- Use express mode to load data that contains the **cdrserver** and **cdrtime** columns. You must perform a level-0 archive after completion.

You can also use deluxe mode without replication to load data. After a deluxe mode load, you do not need to perform a level-0 archive. Deluxe mode also allows you to load TEXT and BYTE data and opaque user-defined types.

For information about HPL, refer to the *Guide to the High-Performance Loader.*

onunload and onload Utilities

You can use the **onunload** and **onload** utilities only to unload and load an entire table. If you want to unload selected columns of a table, you must use either **unload** or the HPL.

*Important: You can only use **onunload** and **onload** in identical (homogeneous) environments.*

For more information about **onunload** and **onload**, see the *Informix Migration Guide.*

dbexport and dbimport Utilities

If you need to copy an entire database for replication, you can use the **dbexport** and **dbimport** utilities. These utilities unload an entire database, including its schema, and then re-create the database. If you want to move selected tables or selected columns of a table, you must use some other utility.

For more information about **dbexport** and **dbimport**, see the *Informix Migration Guide.*

UNLOAD and LOAD Statements

The UNLOAD and LOAD statements allow you to move data within the context of an SQL program.

If the table contains shadow columns, you must:

- Include the **cdrserver** and **cdrtime** columns explicitly in your statement when you unload your data.
- List the columns that you want to load in the INSERT statement and explicitly include the **cdrserver** and **cdrtime** columns in the list when you load your data.

For more information about the UNLOAD and LOAD statements, see the *Informix Guide to SQL: Syntax.*

Data Preparation Example

The following steps provide an example of how to add a new participant (**delta**) to an existing replicate using the LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION statements. Replicate **zebra** replicates data from table **table1** for the following database servers: **alpha**, **beta**, and **gamma**.

The servers **alpha**, **beta**, and **gamma** belong to the server groups **g_alpha**, **g_beta**, and **g_gamma**, respectively. Assume that **alpha** is the database server from which you want to get the initial copy of the data.

To add a new participant to an existing replicate

1.   Declare server **delta** to Enterprise Replication. For example,

```
cdr def ser -c delta -I -S g_alpha g_delta
```

At the end of this step, all servers in the replication environment include information in the **syscdr** database about **delta**, and **delta** has information about all other servers.

2.   Add **delta** as a participant to replicate **zebra**. For example,

```
cdr cha rep -a zebra "dbname@g_delta:owner.table1"
```

This step updates the replication catalog. The servers **alpha**, **beta**, and **gamma** do not queue any qualifying replication data for **delta** because the replicate on **delta**, although defined, has not been started.

3.   Suspend server to **delta** on **alpha**, **beta**, and **gamma**.

```
cdr sus ser g_alpha g_beta g_gamma
```

As a result of this step, replication data is queued for **delta**, but no data is delivered.

4.   Start replication for replicate **zebra** on **delta**.

```
cdr sta rep zebra g_delta
```

This step causes servers **alpha**, **beta**, and **gamma** to start queueing data for **delta**. No data is delivered to **delta** because **delta** is suspended. Then, **delta** queues and delivers qualifying data (if any) to the other servers.

Informix recommends that you do not run any transactions on **delta** that affect table **table1** until you finish the synchronization process.

5.   Unload data from table **table1** using the UNLOAD statement or the **unload** utility on HPL.

6.   Copy the unloaded data to **delta**.

7.   Start transactions with BEGIN WORK WITHOUT REPLICATION, load the data using the LOAD statement, and commit the transaction.

If you used the HPL to unload the data in step 5, then use the HPL Deluxe load without replication to load the data into **table1** on **delta**.

8. Resume server **delta** on **alpha**, **beta**, and **gamma**.

   This step starts the flow of data from **alpha**, **beta**, and **gamma** to **delta**.

   At this point you might see some transactions aborted because of conflict. Transaction can abort because **alpha**, **beta**, and **gamma** started queueing data from **delta** in step 4. However, those same transactions might have been moved in steps 5 and 7.

It might seem strange to declare replication on server **delta** and then immediately suspend replication. You must do this because, while you are preparing the replicates and unloading and loading files, the other servers in the replicate (**alpha**, **beta**, and **gamma**) might be collecting information that needs to be replicated. After you finish loading the initial data to **delta** and resume replication, the information that was generated during the loading process can be replicated.

Defining and Modifying Replication Servers, Replicates, and Participants

In This Chapter

This chapter describes the steps for declaring a database server for Enterprise Replication.

To define a replication server

1. Initialize the database server.

    For information, see "Initializing Database Servers" on page 5-4.

2. Declare the database server to Enterprise Replication.

3. For information, see "Defining Replication Servers" on page 5-4.

    Once you define the server for replication, the server is known as a *replication server*.

4. Define replicates.

    The *replicate* definition includes information about the participants, replication options, frequency, and conflict-resolution rules and scope.

    For information, see "Defining Replicates" on page 5-7.

5. Define participants.

    A *participant* definition specifies the data (database, table, and columns) that should be replicated. Although you can define a replicate with fewer, a replicate should contain two or more participants to be useful.

    For information, see "Defining Participants" on page 5-8.

Important: You must be the Enterprise Replication server administrator ("Enterprise Replication Server Administrator" on page 2-4) to define the replication server.

This chapter also covers the following topics:

- Modifying Replication Servers
- Modifying Replicates
- Resynchronizing Replication Servers

For information on managing your replication servers and replicates, see "Managing Replication Servers and Replicates" on page 6-1.

Initializing Database Servers

The database server must be online before you can declare it for replication.

To bring the server from offline to online, issue the following command for your operating system.

Operating System	Command
UNIX	**oninit**
Windows	**starts** *dbservername*

To bring the server from quiescent mode to online on either UNIX or Windows, enter **onmode** -**m**.

For more information on initializing the database server, see the chapter on database operating modes in the *Administrator's Guide for Informix Dynamic Server*.

Defining Replication Servers

Once you bring the database server online, the next step is to declare the server to Enterprise Replication.

To define the replication server, use the **cdr define server** command. For example:

```
cdr define server --init [--connect=server_name] \
    options server_group_name
```

The --**init** option initializes the server. If INFORMIXSERVER is not set to the server that you are defining, specify the --**connect=***server_name* option ("Connecting to Another Replication Server" on page 6-4).

*Tip: All Enterprise Replication commands and options (except the --**connect** option which can use both) use the name of the database server group (also known as a server group) instead of the more familiar database server name for all references to database servers. This manual uses the convention that the name of a server group is **g_** followed by the server group name.*

Important: If the CDR_SBSPACE configuration parameter is not set in ONCONFIG or specifies an invalid sbspace, Enterprise Replication fails to define the server.

For more information, see "Row Data sbspace" on page 4-13.

Customizing the Replication Server Definition

When you define a replication server, you can change the following aspects of the server in the replication environment:

- Synchronize the new server's global catalog ("Global Catalog" on page 2-6) with another Enterprise Replication server.

    For all servers except the first server you define in your Enterprise Replication system, you must use --**sync=***sync_server* in your server definition to synchronize the global catalog with an existing server.

    For Hierarchical Routing (HR) topologies, Enterprise Replication also uses the synchronization server as the parent of the new server in the current topology. For example, if you add a new server **kauai** and synchronize its global catalog with **hawaii**, **kauai** is connected to **hawaii** in the Enterprise Replication topology. For more information, see "Choosing a Replication Network Topology" on page 3-19.

- Set the idle timeout.

    To specify the time (in minutes) that you want to allow the connection between two Enterprise Replication servers to remain idle before disconnecting, use --**idle=***timeout.*

Depending on the type of Enterprise Replication system (primary-target or update-anywhere) and network topology (fully connected or hierarchical) that you chose in Chapter 3, "Selecting the Enterprise Replication System and Network Topology," set the following options:

- Specify the location of the ATS directory.

    To use ATS, specify the directory for the Aborted Transaction Spooling (ATS) files for the server using --**ats=***dir*.

    For more information, see "Creating ATS and RIS Directories" on page 4-15 and Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."

- Specify the location of the RIS directory.

    To use RIS, specify the directory for the Row Information Spooling (RIS) files for the server using --**ris=***dir*.

    For more information, see "Creating ATS and RIS Directories" on page 4-15 and Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."

- Specify the type of server (hierarchical replication).
  - ❑ To specify the server as a nonroot server, use --**nonroot.**
  - ❑ To specify the server as a leaf server, use --**leaf**.

  If neither --**leaf** nor --**nonroot** is specified, the server is defined as a root server.

  For more information, see "Hierarchical Replication Topologies" on page 3-21.

For more information, see "cdr define server" on page A-20.

Defining Replicates

To define a replicate, use the **cdr define replicate** command.

You must provide the following information in the replicate definition:

- Participants
  See "Defining Participants" on page 5-8.
- Conflict resolution rules and scope
  See "Specifying Conflict Resolution Rules and Scope" on page 5-10.
- Replication frequency
  See "Specifying Replication Frequency" on page 5-10.
- Error logging
  See "Setting Up Error Logging" on page 5-11.
- Replicate full rows or only changed columns
  See "Replicating Only Changed Columns" on page 5-11.
- IEEE or canonical message formats
  See "Using the IEEE Floating Point or Canonical Format" on page 5-13.
- Database triggers
  See "Enabling Triggers" on page 13.

Once you define the replicate and participants, you must manually start the replicate using the **cdr start replicate** command. See "Starting a Replicate" on page 6-7.

Defining Participants

You must define participants for each server involved in the replicate in the replicate definition using the **cdr define replicate** command.

Important: You cannot start and stop replicates that have no participants.

Each participant definition includes the following information:

- Database server group name
- Database in which the table to be replicated resides
- Table name
- Table owner

   See "Table Owner" on page A-89.

- SELECT statement and optional WHERE clause

   See "Participant Modifier" on page A-90.

   If you use SELECT * from *table_name*, the tables must be identical on all database servers defined for the replicate.

Important: Do not create more than one participant definition for each row and column to replicate. If the participant is the same, Enterprise Replication attempts to insert or update duplicate values during replication. For example, if one participant modifier includes WHERE x < 50 and another includes WHERE x < 100, Enterprise Replication sends the data twice.

In addition, for a primary-target replication system, you can specify the participant type as either *primary* or *target* (receive-only). If you do not specify the participant type, Enterprise Replication defines the participant as update-anywhere by default. For more information, see "Primary-Target Replication System" on page 3-3 and "Participant Type" on page A-89.

For example, in the following participant definition, the P indicates that in this replicate, **hawaii** is a primary server.

```
"P db1@g_hawaii:informix.mfct" "select * from mfct" \
```

If any data in the selected columns changes, that changed data is to be sent to the secondary servers.

In the following example, the R indicates that in this replicate, **maui** is a secondary server:

```
"R db2@g_maui:informix.mfct" "select * from mfct"
```

The specified table and columns receive information sent from the primary server. Changes to those columns on **maui** are *not* replicated.

*Important: The **R** in the participant definition indicates that the table is receive-only mode, not that the table is in read-only mode.*

If you do not specify the participant type, Enterprise Replication defines the participant as update-anywhere by default. For example:

```
"db1@g_hawaii:informix.mfct" "select * from mfct" \
"db2@g_maui:informix.mfct" "select * from mfct"
```

For more information, see "Participant" on page A-88.

Defining Replicates on Table Hierarchies

When you define replicates on inherited table hierarchies, use the following guidelines to replicate operations:

- For both the parent and child tables, define a replicate on each table.
- For only the parent table (not the child table), define a replicate on the parent table only.
- For only the child table (not the parent table), define a replicate on the child table only.

Specifying Conflict Resolution Rules and Scope

For update-anywhere replication systems, you must specify the conflict-resolution rules in the replicate definition using the --**conflict**=*rule* option. The conflict resolution rules are:

- **ignore**
- **timestamp**
- *routine_name*

  If you use an SPL routine for your conflict-resolution rule, you can also use the --**optimize** option to specify that the routine is optimized.

For more information, see the following:

- "Update-Anywhere Replication System" on page 3-10
- "Conflict-Resolution Rule" on page 3-12
- "Conflict Options" on page A-11

You can also specify the scope using the --**scope**=*scope* option:

- **transaction** (default)
- **row**

For more information, see "Scope" on page 3-18, and "Scope Options" on page A-11.

Specifying Replication Frequency

The replication frequency options allow you to specify the interval between replications, or the time of day when an action should occur. If you do not specify the frequency, the default action is that replication always occurs immediately when data arrives.

The frequency options are:

- --**immed**
- --**every**=*interval*
- --**at**=*time*

For more information, see "Frequency Options" on page A-92.

Important: If you use time-based replication and two tables have referential constraints, the replicates must belong to the same exclusive replicate set. For more information, see "Exclusive Replicate Sets" on page 7-4.

Setting Up Error Logging

The Aborted Transaction Spooling (ATS) files and Row Information Spooling (RIS) files contain information about failed transactions and aborted rows. You can use this information to help you diagnose problems that arise during replication. Informix recommends that, until you become thoroughly familiar with the behavior of the replication system, you select both options. For more information about these files, refer to Chapter **8**, "Monitoring and Troubleshooting Enterprise Replication."

To configure your replicate to use ATS and RIS

1. First, set up the ATS and RIS directories ("Creating ATS and RIS Directories" on page 4-15).
2. Then, when you define your server ("Defining Replication Servers" on page 5-4), specify the location of the ATS and RIS directories.
3. Now, when you define the replicate, specify that the replicate use ATS and RIS by including the --**ats** and --**ris** options in the replicate definition.

For more information, see "Special Options" on page A-13.

Replicating Only Changed Columns

By default, Enterprise Replication replicates the entire row, even if only one column changed. (For more information on how Enterprise Replication evaluates data for replication, see "Evaluate Data for Replication" on page 1-9.)

You can change the default behavior to replicate only the columns that changed. To replicate only changed columns, include the --**fullrow n** option in the replicate definition.

Tip: *Enterprise Replication always sends the primary key column, even if you specify to replicate only changed columns.*

Replicating only the columns that changed has the following advantages:

- Sends less data

   This is particularly useful when replicating large objects, such as data stored as smart large objects and opaque user-defined types (UDTs). For example, when you replicate a row with a large column that changes infrequently and a small column that changes frequently, Enterprise Replication sends significantly less data each time it updates the row if it only replicates the changed columns.

- Uses fewer Enterprise Replication resources, such as memory

If Enterprise Replication replicates an entire row from the source and the corresponding row does not exist on the target, Enterprise Replication applies the update as an INSERT, also known as an *UPSERT,* on the target. By replicating the entire row, Enterprise Replication self-corrects any errors during replication. If any errors occur in an update of the target database server (for example, a large object is deleted before Enterprise Replication can send the data), the next update from the source database server (a complete row image) corrects the data on the target server.

Important: *If you replicate only changed columns:*

- *Enterprise Replication does not apply upserts.*

   If the row to replicate does not exist on the target, Enterprise Replication does not apply it. If you set up error logging ("Setting Up Error Logging" on page 5-11), Enterprise Replication logs this information as a failed operation.

- *You cannot use the SPL routine or time stamp with SPL routine conflict-resolution rules. For more information, see "Conflict Resolution" on page 3-11.*

- *All database servers in the enterprise must be Version 9.3 or later.*

   Enterprise Replication does not enforce this restriction. If you attempt to replicate only changed columns to a pre-Version 9.3 database server, you will experience corrupt data on that database server.

Enterprise Replication logs bitmap information about the updated columns in the logical-log file. For more information, see the CDR record type in the logical-logs chapter in the *Informix Administrator's Reference.*

For more information on the --**fullrow** option, see "Special Options" on page A-13.

Using the IEEE Floating Point or Canonical Format

The FLOAT and SMALLFLOAT data types are handled inconsistently across different platforms. You can specify sending this data in either IEEE floating point format or machine-independent decimal representation:

- Enable IEEE floating point format to send all floating point values in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating point format.

  To use IEE floating point format, include the --**floatieee** option in your replicate definition.

  Informix recommends that you define all new replicates with the --**floatieee** option.

- Enable canonical format to send floating-point values in a machine-independent decimal representation when you replicate data between dissimilar hardware platforms.

  To use canonical format, include the --**floatcanon** option in your replicate definition. The --**floatcanon** option is provided for backward-compatibility only; Informix recommends that you use the --**floatieee** option when defining new replicates.

- If you specify neither IEEE or canonical formats, Enterprise Replication sends FLOAT and SMALLFLOAT data types as a straight copy of machine representation. If you are replicating across different platforms, replicated floating-point numbers will be incorrect.

For more information, see "Special Options" on page A-13.

*Important: Once set, you cannot modify the replicate to change the --**floatieee** or --**floatcanon** options.*

Enabling Triggers

By default, when a replicate causes an insert, update, or delete on a target table, triggers associated with the table are not executed. However, you can specify that triggers are executed when the replicate data is applied by enabling triggers in the replicate definition.

To enable triggers, include the --**firetrigger** option in your replicate definition.

For information, refer to "Triggers" on page 2-13 and "Special Options" on page A-13.

Modifying Replication Servers

To modify a replication server, use the **cdr modify server** command. With this command, you can change the following attributes of the server:

- Idle timeout
- Location of the directory for the Aborted Transaction Spooling (ATS) files
- Location of the directory for the Row Information Spooling (RIS) files

For information about each of these attributes, see "Defining Replication Servers" on page 5-4. For more information, see "cdr modify server" on page A-56.

Modifying Replicates

You can modify replicates in two ways:

- Adding or Deleting Participants
- Changing Replicate Attributes

Adding or Deleting Participants

To be useful, a replicate must include at least two participants. You can define a replicate ("Defining Replicates" on page 5-7) that has fewer than two participants, but before you can use that replicate, you must add more participants.

To add a participant to an existing replicate, use **cdr change replicate** --**add**. For example, to add two participants to the **sales_data** replicate, enter:

```
cdr change replicate --add sales_data \
    "db1@hawaii:jane.table1" "select * from table1" \
    "db2@maui:john.table2" "select * from table2"
```

To delete a participant from the replicate, use **cdr change replicate** --**delete**.

For example, to delete these two participants from the replicate, enter:

```
cdr change replicate --delete sales_data \
    "db1@hawaii:jane.table1" "db2@maui:john.table2"
```

For more information, see "cdr change replicate" on page A-4.

Changing Replicate Attributes

You can change the following attributes of a replicate using the **cdr modify replicate** command:

- Conflict-resolution rules and scope
- Replication frequency
- Error logging
- Replicate full rows or only changed columns
- Database triggers
- Participant type

Important: *You cannot change the conflict resolution from ignore to a non-ignore option (time stamp, SPL routine, or time stamp and SPL routine). You cannot change a non-ignore conflict resolution option to ignore.*

For information on each of these attributes, see "Defining Replicates" on page 5-7.

For example, to change the replication frequency for the **sales_data** replicate to every Sunday at noon, enter:

```
cdr modify replicate sales_data Sunday.12:00
```

For more information, see "cdr modify replicate" on page A-49.

Resynchronizing Replication Servers

You might want to resynchronize servers in your replication environment, for example, if replicated tables between Enterprise Replication servers have become out of sync after stopping and restarting replication for a server.

Warning: Make sure that there are no user transactions performing updates on any replicated tables while you run this procedure.

To synchronize the replication g_papeete with the g_raratonga

1. Suspend replication to the replication server group **g_papeete.**

   See "Suspending Replication for a Server" on page 6-5.

2. Unload the table from the server group **g_raratonga.**

   See "Loading and Unloading Data" on page 4-22.

3. Load the table on **g_papeete** and specify BEGIN WORK WITHOUT REPLICATION.

   See "Loading and Unloading Data" on page 4-22 and "Blocking Replication" on page 4-18.

4. Resume replication to **g_papeete**.

   See "Resuming a Suspended Replication Server" on page 6-6.

Important: If tables that you are synchronizing include shadow columns, you must explicitly unload and load these columns. If these values are not included, Enterprise Replication inserts NULL values. For more information, see "Shadow Column Disk Space" on page 4-11 and "Loading and Unloading Data" on page 4-22.

Managing Replication Servers and Replicates

In This Chapter

This chapter covers how to manage your Enterprise Replication system, including managing replication servers, replicates and participants in Enterprise Replication, and replication server network connections.

Managing Replication Servers

The *state* of the server refers to the relationship between the source server and the target server. To determine the current state of the server, use **cdr list server** *server_name*. For more information about the possible server states, see "The STATE Column" on page A-46.

Viewing Replication Server Attributes

Once you define a server for replication ("Defining Replication Servers" on page 5-5), you can view information about the server using the **cdr list server** command. If you do not specify the name of a defined server on the command line, Enterprise Replication lists all the servers that are visible to the current server. If you specify a server name, Enterprise Replication displays information about the current server, including server ID, server state, and attributes.

For more information, see "cdr list server" on page A-45.

Connecting to Another Replication Server

By default, when you view information about a server, Enterprise Replication connects to the global catalog of the database server specified by **$INFORMIXSERVER**. You can connect to the global catalog of another database server using the --**connect** option.

For example, to connect to the global catalog of the database server **idaho**, enter:

```
cdr list server --connect idaho
```

For more information, see "Global Catalog" on page 2-6 and "Connect Option" on page A-87.

Stopping Replication on a Server

Enterprise Replication starts automatically when you use **cdr define server** to declare the server for replication. The replication threads continue running until you shut the database server down or you remove the server with **cdr delete server**. If you shut down the database server while Enterprise Replication is running, replication begins again when you restart the database server. For more information about managing the database server, see the chapter on database operating modes in the *Administrator's Guide for Informix Dynamic Server*.

Generally, to stop Enterprise Replication on a server, you should bring down the server. However, you might want to temporarily stop the Enterprise Replication threads without stopping the server. To stop replication, use the **cdr stop** command.

When you use **cdr stop**, Enterprise Replication stops capturing data to be replicated. To ensure consistency, verify that no database update activity occurs while Enterprise Replication is stopped. Replication threads remain stopped (even if the database server is stopped and restarted) until you issue a **cdr start** command.

Warning: When you stop replication on a server, you must ensure that the send queues on the other Enterprise Replication servers participating in replication do not fill.

For more information, see "cdr stop" on page A-71.

Restarting Replication on a Stopped Server

To restart replication for a stopped server, use the **cdr start** command. When you restart the server, the Enterprise Replication threads start and continue.

Enterprise Replication resumes evaluating the logical logs at the replay position (where Enterprise Replication stopped evaluating the logical log when the server was stopped).

Warning: If the logical log corresponding to the replay position no longer exists, then the restart partially fails and no database updates performed on the local database server are replicated.

For more information, see "cdr start" on page A-65.

Suspending Replication for a Server

If you do not want to completely shut down the Enterprise Replication threads, you can suspend replication of data to the server using the **cdr suspend server** command. When replication is suspended to the server, the source server queues replicated data, but suspends delivery of replicated data to the target server. Note that this command does not affect the network connection to the suspended server. The source server continues to send other messages, such as acknowledgement and control messages.

For example, to suspend replication of data to the server group **g_papeete** from the server group **g_raratonga**, enter:

```
cdr suspend server g_papeete g_raratonga
```

To suspend replication to **g_papeete** from all servers in the enterprise, enter:

```
cdr suspend server g_papeete
```

Warning: When you suspend replication on a server, you must ensure that the send queues on the other Enterprise Replication servers participating in replication do not fill.

For more information, see "cdr suspend server" on page A-81.

Resuming a Suspended Replication Server

To resume replication to a suspended server, use the **cdr resume server** command. When you resume the server, the queued data is delivered.

For example, to resume replication to the **g_papeete** server group, enter:

```
cdr resume server g_papeete
```

If you do not specify the server group (**g_papeete** in the example), **cdr resume server** resumes replication from all servers participating in replication.

For more information, see "cdr resume server" on page A-63.

Deleting a Replication Server

To delete a replication server, use the **cdr delete server** command. You must run the **cdr delete server** command twice.

For example, to remove the server group **g_papeete** from Enterprise Replication, set **$INFORMIXSERVER** to **papeete** and run the following commands:

```
cdr delete server g_papeete
cdr delete server --connect raratonga g_papeete
```

The first command deletes the local server group (**g_papeete**) from Enterprise Replication and the second connects to another server in the replication environment (**raratonga**) and deletes **g_papeete** from that server. The change then replicates to the other servers in the replication environment.

Warning: *Avoid deleting an Enterprise Replication server and immediately recreating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), you might experience failures in the Enterprise Replication system at the time of the operation or later. For more information, see "Operational Considerations" on page 2-8.*

For more information, see "cdr delete server" on page A-29.

Managing Replicates

For more information about the possible replicate states, see "Displaying Information About Replicates" on page A-40.

Important: If a replicate belongs to an exclusive replicate set ("Exclusive Replicate Sets" on page 7-4), you cannot manage the replicates individually. With the exception of deleting the replicate, you must manage the replicate as part of the exclusive replicate set. For more information, see "Managing Replicate Sets" on page 7-6.

If the replicate belongs to a non-exclusive replicate set ("Non-Exclusive Replicate Sets" on page 7-5), you can manage the replicates both individually and as part of the set.

Viewing Replicate Properties

Once you define a replicate ("Defining Replicates" on page 5-7), you can view the properties of the replicate using the **cdr list replicate** command. If you do not specify the name of a defined replicate on the command line, Enterprise Replication lists all the replicates defined on the current server. If you specify a replicate name, Enterprise Replication displays detailed information, including participant information, about the replicate.

For information about this command, see "cdr list replicate" on page A-39.

Starting a Replicate

When you define a replicate, the replicate does not begin until you explicitly change its state to *active*. When a replicate is active, Enterprise Replication captures data from the logical log and transmits it to the participants.

Important: You cannot start replicates that have no participants.

To change the replicate state to active, use the **cdr start replicate** command. For example, to start the replicate **sales_data** on the servers **server1** and **server23**, enter:

```
cdr start replicate sales_data server1 server23
```

This command causes **server1** and **server23** to start sending data for the **sales_data** replicate.

If you omit the server names, this command starts the replicate on all servers that are included in that replicate.

*Warning: Run the **cdr start replicate** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.*

Important: If a replicate belongs to an exclusive replicate set ("Exclusive Replicate Sets" on page 7-4), you must start the replicate set to which the replicate belongs. For more information, see "Starting a Replicate Set" on page 7-7.

For more information, see "cdr start replicate" on page A-67.

Stopping a Replicate

To stop the replicate, use the **cdr stop replicate** command. This command changes the replicate state to *inactive* and deletes any data in the send queue for that replicate. When a replicate is inactive, Enterprise Replication does not capture, transmit, or process any database changes.

Important: You cannot stop replicates that have no participants.

For example, to stop the **sales_data** replicate on the servers **server1** and **server23**, enter:

```
cdr stop replicate sales_data server1 server23
```

This command causes **server1** and **server23** to purge any data in the send queue for the **sales_data** replicate and stops sending data for that replicate. Any servers not listed on the command line continue to capture and send data for the **sales_data** replicate (even to **server1** and **server23**).

If you omit the server names, this command stops the replicate on all servers that are included in that replicate.

Important: If a replicate belongs to an exclusive replicate set ("Exclusive Replicate Sets" on page 7-4), you must stop the replicate set to which the replicate belongs. For more information, see "Stopping a Replicate Set" on page 7-7.

For more information, see "cdr stop replicate" on page A-73.

Suspending a Replicate

If you do not want to completely halt all processing for a replicate, you can suspend a replicate using the **cdr suspend replicate** command. When a replicate is suspended, the replicate captures and accumulates changes to the source database, but does not transmit the captured data to the target database.

For example, to suspend the **sales_data** replicate, enter:

```
cdr suspend replicate sales_data
```

Important: *If a replicate belongs to an exclusive replicate set ("Exclusive Replicate Sets" on page 7-4), you must suspend the replicate set to which the replicate belongs. For more information, see "Suspending a Replicate Set" on page 7-7.*

For more information, see "cdr suspend replicate" on page A-77.

Resuming a Suspended Replicate

To return the state of a suspended replicate to active, use the **cdr resume replicate** command. For example:

```
cdr resume replicate sales_data
```

Important: *If a replicate belongs to an exclusive replicate set ("Exclusive Replicate Sets" on page 7-4), you must resume the replicate set to which the replicate belongs. For more information, see "Resuming a Replicate Set" on page 7-8.*

For more information, see "cdr resume replicate" on page A-59.

Deleting a Replicate

To delete the replicate from the global catalog, use the **cdr delete replicate** command. When you delete a replicate, Enterprise Replication purges all replication data for the replicate from the send queue at all participating database servers.

For example, to delete **sales_data** from the global catalog, enter:

```
cdr delete replicate sales_data
```

> ⚠️ **Warning:** *Avoid deleting a replicate and immediately re-creating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), you might experience failures in the Enterprise Replication system at the time of the operation or later. For more information, see "Operational Considerations" on page 2-8.*

For more information, see "cdr delete replicate" on page A-25.

Managing Replication Server Network Connections

Managing replication network connections consists of viewing the network connection status, dropping the network connection, and reestablish a dropped network connection.

Viewing Network Connection Status

To determine the current status of the network connection to each of the servers participating in replication, use **cdr list server**. For more information about network connection status, see "The STATUS Column" on page A-46.

For more information, see "cdr list server" on page A-45.

Dropping the Network Connection

To drop the Enterprise Replication network connection for a server, use the **cdr disconnect server** command. When you drop the connection, Enterprise Replication continues to function and queue transactions. For example, to disconnect the network connection between the current replication server and the server **g_papeete**, enter:

```
cdr disconnect server g_papeete
```

> ⚠️ **Warning:** *When you disconnect a server from Enterprise Replication, you must ensure that the send queues on **all** other Enterprise Replication servers participating in replication do not fill.*

For more information, see "cdr disconnect server" on page A-33.

Reestablishing the Network Connection

To reestablish a dropped network connection, use the **cdr connect server** command.

For example, to reestablish the network connection between the current replication server and the server **g_papeete**, enter:

```
cdr connect server g_papeete
```

For more information, see "cdr connect server" on page A-9.

Creating and Managing Replicate Sets

In This Chapter

This chapter covers creating replicate sets, viewing replicate set properties, and managing and modifying replicate sets.

A *replicate set* is a collection of several replicates that you can manage together as a unit. By default, you can manage replicates that belong to a replicate set both individually and as part of the set, or as a set only.

Important: Enterprise Replication supports replicate sets for Informix Dynamic Server Version 9.3 and later only. You cannot define or modify replicate sets to include replicates with participants that are Version 9.2 and earlier. In addition, replicate sets are different from and incompatible with replicate groups (in Version 9.2 and earlier).

To create, manage, or modify a replicate set on a database server other than the server specified by **$INFORMIXSERVER**, use the --**connect** option ("Connecting to Another Replication Server" on page 6-4) to connect to that server.

Creating Replicate Sets

To create a replicate set, use the **cdr define replicateset** command.

Enterprise Replication supports two types of replicate sets: *exclusive* and *non-exclusive.*

Exclusive Replicate Sets

To maintain referential integrity between replicates that include tables that have referential constraints placed on columns, you must create an *exclusive replicate set* and add the replicate to it.

An exclusive replicate set has the following characteristics:

- All replicates in an exclusive replicate set have the same state and frequency ("Displaying Information About Replicates" on page A-40) settings.
- When you create the replicate set, Enterprise Replication sets the initial state of the replicate set to active.
- You can manage the replicates in an exclusive replicate set only as part of the set. Enterprise Replication does not support the following actions for the individual replicates in an exclusive replicate set:
  - ❑ Starting a Replicate
  - ❑ Stopping a Replicate
  - ❑ Suspending a Replicate
  - ❑ Resuming a Suspended Replicate
- Replicates that belong to an exclusive replicate set cannot belong to any other replicate sets.

To create an exclusive replicate set, use the --**exclusive** (-**X**) option with **cdr define replicateset**.

Important: Once you create an exclusive replicate set, you cannot change it to non-exclusive and vice versa.

Non-Exclusive Replicate Sets

By default, **cdr define replicateset** creates *non-exclusive* replicate sets.

A non-exclusive replicate set has the following characteristics:

- You can manage replicates that belong to a non-exclusive replicate set both individually and as part of the set.
- Because individual replicates in a non-exclusive replicate set can have different states, the non-exclusive replicate set itself has no state.
- You should not use non-exclusive replicate sets for replicates that include tables that have referential constraints placed on columns.
- A replicate can belong to more than one non-exclusive replicate set.

Important: Once you create a non-exclusive replicate set, you cannot change it to exclusive.

Use non-exclusive replicate sets if you want to add a replicate to more than one replicate set. For example, you might want to create replicate sets to manage replicates on the target server, table, or entire database. To do this, create three non-exclusive replicate sets:

- A set that contains the replicates that replicate to the target server
- A set that contains the replicates on a particular table
- A set that contains all the replicates

In this scenario, each replicate belongs to three non-exclusive replicate sets.

Customizing the Replicate Set Definition

You can specify the replication frequency ("Specifying Replication Frequency" on page 5-10) for all the replicates when you define the replicate set. For example, to define the non-exclusive replicate set **sales_set** with the replicates **sales_fiji** and **sales_tahiti** and specify that the members of **sales_set** replicate at 4:00 A.M. every day, enter:

```
cdr define replicateset --at 4:00 sales_set sales_fiji \
    sales_tahiti
```

To define the exclusive replicate set **dev_set** with the replicates **dev_pdx** and **dev_lenexa** and specify that the members of **dev_set** replicate at 5:00 P.M. every day, enter:

```
cdr define replicateset -X --at 17:00 dev_set dev_pdx\
    dev_lenexa
```

Important: For replicates that belong to an exclusive replicate set, you cannot specify the frequency for the individual replicates in the set.

For more information, see "cdr define replicateset" on page A-17.

Viewing Replicate Set Properties

To view the properties of the replicate set, use the **cdr list replicateset** command. The **cdr list replicateset** command displays the replicate set name and a list of the replicates that are members of the set. To find out more about each replicate in the replicate set, see "Viewing Replicate Properties" on page 6-7.

For more information, see "cdr list replicateset" on page A-43.

Managing Replicate Sets

When you create a replicate set, you can manage the replicates that belong to that set together or individually. If the replicate set is exclusive, you can only manage the individual replicates as part of the set.

*Important: All operations on a replicate set (except **cdr delete replicateset**) are equivalent to performing the operation on each replicate in the replicate set individually.*

For more information, see "Managing Replicates" on page 6-7.

Starting a Replicate Set

To change the state of all the replicates in the replicate set to active, use the **cdr start replicateset** command. For example, to start the replicate set **sales_set**, enter:

```
cdr start replicateset sales_set
```

*Warning: Run the **cdr start replicateset** command on an idle system (when no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.*

For more information, see "cdr start replicateset" on page A-69 and "cdr start replicate" on page A-67.

Stopping a Replicate Set

To stop the replicates in the replicate set, use the **cdr stop replicateset** command. This command changes the state of all the replicates in the set to inactive.

For example, to stop the **sales_set** replicate set, enter:

```
cdr stop replicateset sales_set
```

For more information, see "cdr stop replicateset" on page A-75 and "cdr stop replicate" on page A-73.

Suspending a Replicate Set

If you do not want to completely halt all processing for the replicates in a replicate set, you can suspend the replicates in the set using the **cdr suspend replicateset** command.

For example, to suspend the **sales_set** replicate set, enter:

```
cdr suspend replicateset sales_set
```

For more information, see "cdr suspend replicateset" on page A-79 and "cdr suspend replicate" on page A-77.

Resuming a Replicate Set

To return the suspended replicates in the replicate set to active, use the **cdr resume replicateset** command. For example:

```
cdr resume replicateset sales_set
```

For more information, see "cdr resume replicateset" on page A-61 and "cdr resume replicate" on page A-59.

Deleting a Replicate Set

To delete the replicate set, use the **cdr delete replicateset** command.

Tip: *When you delete a replicate set, Enterprise Replication does not delete the replicates that are members of the replicate set. The replicates remain in the state they were in when the set was deleted.*

For example, to delete **sales_set**, enter:

```
cdr delete replicateset sales_set
```

Warning: *Avoid deleting a replicate set and immediately re-creating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), you might experience failures in the Enterprise Replication system at the time of the operation or later. For more information, see "Operational Considerations" on page 2-8.*

For more information, see "cdr delete replicateset" on page A-27.

Modifying Replicate Sets

You can modify replicate sets in two ways:

- Adding or Deleting Replicates From a Replicate Set
- Changing Replication Frequency For the Replicate Set

Adding or Deleting Replicates From a Replicate Set

To add a replicate to an existing replicate set, use the command **cdr change replicateset** --**add**. For example, to add two replicates to **sales_set**, enter:

```
cdr change replicateset --add sales_set sales_kauai \
    sales_moorea
```

When you add a replicate to:

- A non-exclusive replicate set, the state of the new replicate remains as it was when you added it to the set. To bring all the replicates in the non-exclusive set to the same state, use one of the commands described in "Managing Replicate Sets" on page 7-6.
- An exclusive replicate set, Enterprise Replication changes the existing state and replication frequency settings of the replicate to the current properties of the exclusive replicate set.

To delete a replicate from the replicate set, use **cdr change replicate** --**delete**.

For example, to delete the two replicates, **sales_kauai** and **sales_moorea**, from the replicate set, enter:

```
cdr change replicateset --delete sales_set sales_kauai\
    sales_moorea
```

When you add or remove a replicate from an exclusive replicate set that is suspended ("Suspending a Replicate Set" on page 7-7) or that is defined with a frequency interval ("Frequency Options" on page A-92), Enterprise Replication transmits all the data in the queue for the replicates in the replicate set up to the point when you added or removed the replicate.

For more information, see "cdr change replicateset" on page A-6.

Changing Replication Frequency For the Replicate Set

You can change the replication frequency ("Specifying Replication Frequency" on page 5-10) for the replicates in an exclusive or non-exclusive replicate set using the **cdr modify replicateset** command.

For example, to change the replication frequency for each of the replicates in the **sales_set** to every Monday at midnight, enter:

```
cdr modify replicateset sales_set Monday.24:00
```

For more information, see "cdr change replicateset" on page A-6.

Monitoring and Troubleshooting Enterprise Replication

In This Chapter

Enterprise Replication provides tools to help diagnose problems that arise during replications. The Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files contain information about failed transactions.

In addition, you can use tools provided with the server, such as the **onstat** command, to display statistics that you can use to diagnose problems. For more information on the **onstat** commands that are relevant to Enterprise Replication, see Appendix C, "onstat Command Reference."

This chapter covers the following topics:

- ■ Aborted Transaction Spooling Files
- ■ Row Information Spooling Files
- ■ Preventing Memory Queues from Overflowing
- ■ Solving Common Configuration Problems
- ■ Enterprise Replication Event Alarms

Aborted Transaction Spooling Files

When ATS is active, all failed replication transactions are recorded in ATS files. Each ATS file contains all the information pertinent to a single failed transaction. If a replicated transaction fails for any reason (constraint violation, duplication, and so on), all the buffers in the replication message that compose the transaction are written to a local operating-system file. You can use the ATS files to identify problems or as input to custom utilities that extract or reapply the aborted rows.

Aborted-transaction spooling only occurs if the entire transaction is aborted. Transactions defined with row scope that have aborted rows but are success-fully committed on the target tables are not logged.

All rows that fail conflict resolution for a transaction that has row scope defined are also written to the RIS file ("Row Information Spooling Files" on page 8-9), if defined.

In some cases, such as with long transactions, the database server itself aborts transactions. In these cases, Enterprise Replication does *not* generate an ATS file.

Preparing to Use ATS

Failed transactions are not automatically recorded in ATS files.

To collect ATS information

1. Create a directory for Enterprise Replication to store ATS files.

   ■ If you are using primary-target replication, create the directory on the target system.

   ■ If you are using update-anywhere replication and have conflict resolution enabled, create the directory on all participating repli-cation systems.

   For more information, see "Creating ATS and RIS Directories" on page 4-15.

2. When you define a server for Enterprise Replication, specify the location of the ATS directory you created in step 1.

   To do this, include the --**ats** option with the **cdr define server** command.

   If you do not specify an ATS directory:

   UNIX

   ■ Enterprise Replication stores the ATS files in the **/tmp** directory. ♦

   WIN NT/2000

   ■ Enterprise Replication stores the ATS files in the **\tmp** directory of the drive that contains **%INFORMIXDIR%** ♦

   For more information, see "Creating ATS and RIS Directories" on page 4-15.

3. When you define a replicate, specify that ATS is active.

   To do this, include the --**ats** option with the **cdr define replicate** command.

   For more information, see "Setting Up Error Logging" on page 5-11.

About ATS Filenames

When ATS is active, each aborted transaction is written to a file in the specified directory.

The following table provides the naming convention for ATS files:

```
ats.target.source.threadId.timestamp.sequence
```

Name	Description
target	The name of the database server receiving this replicate transaction
source	The name of the database server that originated the transaction
threadId	The identifier of the thread that processed this transaction
timestamp	The value of the internal time stamp at the time that this ATS instance was started
sequence	A unique integer, incremented by ATS each time a transaction is spooled

The naming convention ensures that all filenames that ATS generates are unique, and therefore name collision is unlikely. However, when ATS opens a file for writing, any previous file contents will be overwritten. (ATS does not append to a spool file; if a name collision does occur with an existing file, the original contents of the file will be lost.)

The following is an example ATS filename for a transaction sent by server **g_amsterdam** to server **g_beijing**:

```
ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6
```

About ATS File Information

The first three characters in each line of the ATS spool file describe the type of information for the line, as the following table defines.

Label	Name	Description
TXH	Transaction heading	This line contains information from the transaction header, including the sending server ID and the commit time, the receiving server ID and the received time, and any Enterprise Replication, SQL, or ISAM error information for the transaction.
RRH	Replicated row heading	This line contains header information from the replicated rows, including the row number within the transaction, the group ID, the replicate ID (same as replicate group ID if replicate is not part of any replicate group), the database, owner, table name, and the database operation.
RRS	Replicated row shadow columns	This line contains shadow column information from replicated rows, including the source server ID and the time when the row is updated on the source server. This line is printed only if the replicate is defined with a conflict-resolution rule.
RRD	Replicated row data	This line contains the list of replicated columns in the same order as in the SELECT statement in the define replicate command. Each column is separated by a '	' and displayed in ASCII format.

When the spooling program encounters severe errors (for example, cannot retrieve replicate id for the replicated row, unable to determine the replicated column's type, size, or length), it displays this row data in hexadecimal format. The spooling program also displays the row data in hexidecimal format if a row includes replicated UDT columns. |

BYTE and TEXT Information in ATS Files

When the information recorded in the ATS file includes BYTE or TEXT data, the replicated row data (RRD) information is reported, as the following examples show.

Example 1

```
<1200, TEXT, PB 877(necromsv) 840338515(00/08/17 20:21:55)>
```

In this example:

- `1200` is the size of the data.
- `TEXT` is the data type (it is either BYTE or TEXT).
- `PB` is the storage type (PB when the BYTE or TEXT is stored in the tablespace, BB for blobspace storage).
- The next two fields are the server identifier and the time stamp for the column if the conflict-resolution rule is defined for this replicate and the column is stored in a tablespace.

Example 2

```
<500 (NoChange), TEXT, PB 877(necromsv) 840338478(00/08/17 20:21:18)>
```

In this example, (`NoChange`) after the 500 indicates that the TEXT data has a size of 500, but the data has not been changed on the source server. Therefore the data is not sent from the source server.

Example 3

```
<(Keep local blob),75400, BYTE, PB 877(necromsv) 840338515(00/08/17
20:21:55)>")
```

In this example, (`Keep local blob`) indicates that the replicated data for this column was not applied on the target table, but instead the local BYTE data was kept. This usually happens when time-stamp conflict resolution is defined and the local column has a time stamp greater than the replicated column.

Changed Column Information in ATS Files

If you define a replicate to only replicate columns that changed, the RRD entry in the ATS and RIS files show a ? for the value of any columns that are not available. For example:

```
RRD 427|amsterdam|?|?|?|?|?|?|?|?|?|?|?
```

For more information, see "Replicating Only Changed Columns" on page 5-11).

BLOB and CLOB Information in ATS Files

If a replicate includes one or more BLOB or CLOB columns, the RRD entry in the ATS file displays the smart blob metadata (the in-row descriptor of the data), not the smart blob itself, in hexadecimal format in the ATS file. See "UDT Information in ATS Files" for an example.

UDT Information in ATS Files

If a replicate includes one or more UDT columns, the RRD entry in the ATS file displays the row data in hexadecimal format in the ATS file. This representation includes a column number in the form C*nnn* where *nnn* is the column number. For example:

```
RRS C001 0000000a                                 ^@
RRS C002 394925cb                                 ^@9I%Ë
RRD C003 00000016                                 ^@
RRD C004 074b4d70 5d6f784a 00000000 00000000      ^@.KMp]oxJ
```

Row Information Spooling Files

Row Information Spooling logs the following types of information in RIS files:

- Individual aborted row errors
- Replication exceptions (such as when a row is converted by Enterprise Replication from insert to update, or from update to insert, and so on)
- Special SPL routine return codes, as defined by the application (if an SPL routine is called to resolve a conflict)

Preparing to Use RIS

Failed transactions are not automatically recorded in RIS files.

To collect RIS information

1. Create a directory for Enterprise Replication to store RIS files.

    If you have conflict resolution enabled, create the directory on all participating replication systems.

    For more information, see "Creating ATS and RIS Directories" on page 4-15.

2. When you define a server for Enterprise Replication, specify the location of the RIS directory you created in step 1.

    To do this, include the --**ris** option with the **cdr define server** command.

    If you do not specify an RIS directory:

UNIX

WIN NT/2000

- Enterprise Replication stores the RIS files in the /**tmp** directory. ♦
- Enterprise Replication stores the RIS files in the **tmp** directory of the drive that contains **%INFORMIXDIR%**. ♦

    If the default directory does not exist, Enterprise Replication returns an error.

    For more information, see "Creating ATS and RIS Directories" on page 4-15.

3.   When you define a replicate, specify that RIS is active.

To do this, include the --**ris** option with the **cdr define replicate** command.

For more information, see "Setting Up Error Logging" on page 5-11.

About RIS Filenames

The RIS row information is written at the time that the data-synchronization component finishes processing the replicated row and can therefore also provide the local row information. The RIS filename algorithm is analogous to the one that ATS uses, with the *ats* prefix replaced by *ris*. The RIS file that corresponds to the ATS file described in the previous section is, for example:

```
ris.g_beijing.g_amsterdam.D_2.000529_23:27:16.5
```

For more information, see "About ATS Filenames" on page **8**-5.

In addition to the four types of records printed in ATS, the RIS file also includes the following information.

Label	Name	Description
LRH	Local-row header	Indicates if the local row is found in the delete table and not in the target table.
LRS	Local-row shadow columns	Contains the server id and the time when the row is updated on the target server. This line is printed only if the replicate is defined with a conflict-resolution rule.
LRD	Local-row data	Contains the list of replicated columns extracted from the local row and displayed in the same order as the replicated row data. Similar to the replicated row data, each column is separated by a ' | ' and written in ASCII format.
		When the spooling program encounters severe errors (for example, cannot retrieve replicate id for the replicated row, unable to determine the replicated column's type, size, or length) or the table includes UDT columns (whether defined for replication or not), it displays the replicated row data in hexadecimal format. In this case, the local row data is not spooled.

BYTE and TEXT Information in RIS Files

When the information recorded in the RIS file includes BYTE or TEXT data, the RRD information is reported as the following examples show.

Example 1

```
"<1200, TEXT, PB 877(necromsv) 840338515(00/08/17 20:21:55)>
```

In this example:

- ■   `1200` is the size of the TEXT data.
- ■   `TEXT` is the data type (it is either BYTE or TEXT).
- ■   `PB` is the storage type (PB for storage in the tablespace, BB for blobspace storage).
- ■   The next two fields are the server identifier and the time stamp for the column if the conflict-resolution rule is defined for this replicate and the column is stored in a blobspace.

Example 2

```
"<500 (NoChange), TEXT, PB 877(necromsv) 840338478(0000/08/17 20:21:18)>
```

In this example, `(NoChange)` after `500` indicates the TEXT data has size of 500 but the data has not been changed on the source server. Therefore the data is not sent from the source server.

Changed Column Information in RIS Files

See "Changed Column Information in ATS Files" on page 8-8.

BLOB and CLOB Information in RIS Files

See "BLOB and CLOB Information in ATS Files" on page 8-8.

UDT Information in RIS Files

See "UDT Information in ATS Files" on page 8-8.

Preventing Memory Queues from Overflowing

In a well-tuned Enterprise Replication system, the send queue and receive queue should not regularly overflow from memory to disk. However, if the queues in memory fill, the transaction buffers are written (*spooled*) to disk. Spooled transactions consist of *transaction records* and *row data*. Spooled transaction records are stored in the transaction tables in a single dbspace. Spooled row data is stored in a single sbspace.

For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-11.

The following situations can cause Enterprise Replication to spool to disk:

- Receiving server is down or suspended.
- Network connection is down.

   If the receiving server or network connection is down or suspended, Enterprise Replication might spool transaction buffers to disk.

   To check for a down server or network connection, run **cdr list server** on a root server. This command shows all servers and their connection status and state.

   For more information, see "Viewing Replication Server Attributes" on page 6-3 and "cdr list server" on page A-45.

- Replicate is suspended.

   If a replicate is suspended, Enterprise Replication might spool transaction buffers to disk.

   To check for a suspended replicate, run **cdr list replicate**. This command shows all replicates and their state.

   For more information, see "Viewing Replicate Properties" on page 6-7 and "cdr list replicate" on page A-39.

- Enterprise Replication is replicating large transactions.

   Enterprise Replication is optimized to handle small transactions efficiently. Very large transactions or batch jobs may force Enterprise Replication into an exceptional processing path that results in spooling. Avoid replicating these types of transactions.

   For more information, see "Large Transactions" on page 2-15.

- Logical log files are too small or too few.

  If the logical log files are too small or the number of logical log files is too few, Enterprise Replication is more likely to spool transaction buffers to disk.

  To increase the size of the logical logs, see the chapter on logical logs in the *Administrator's Guide for Informix Dynamic Server*. For more information on configuring your logical log files for use with Enterprise Replication, see "Logical Log Configuration Guidelines" on page 4-10.

- Server is overloaded.

  If a server is low on resources, Enterprise Replication might not be able to hold all transactions replicating from a source server in memory during processing, and the transactions spool to disk.

  If this happens, check the system resources; in particular, check disk speed, RAM, and CPU resources.

Preventing Blockout Mode

If the database server comes close to overwriting a logical log that Enterprise Replication has not yet processed (*log wrap*), Enterprise Replication enters *blockout mode*. In blockout mode, user transactions are blocked and you see the following error in the message log:

```
DDR Log Snooping - Catchup phase started, userthreads blocked
```

Blockout mode can occur if your Enterprise Replication system is regularly spooling to disk. To prevent blockout mode, prevent Enterprise Replication from spooling to disk. For information, see "Preventing Memory Queues from Overflowing" on page 8-13.

If Enterprise Replication enters blockout mode, you can immediately increase the size or number of the logical logs (see the chapter on logical logs in the *Administrator's Guide for Informix Dynamic Server*) while you determine what situation is causing excessive spooling. You should also check to see if the Enterprise Replication dbspace or sbspace (for send and receive queue spooling) is running out of disk space (see "Monitoring Disk Usage for Send and Receive Queue Spool" on page 8-15 and "Increasing the Sizes of Storage Spaces" on page 8-15).

Use **onstat** -**g ddr** to monitor the status of the Enterprise Replication database log reader. For more information, see "onstat -g ddr" on page C-4.

If the situation that is causing blockout mode is not resolved, you see the following error in the message log:

```
WARNING:  The replay position was overrun, data may not be replicated.
```

Once this occurs, you must manually resynchronize the source and target databases. For more information, see "Resynchronizing Replication Servers" on page 5-17.

Monitoring Disk Usage for Send and Receive Queue Spool

You should periodically monitor disk usage for the dbspace (specified by CDR_QHDR_DBSPACE configuration parameter) and the sbspace (CDR_QDATA_SBSPACE) that Enterprise Replication uses to spool the queues to disk.

To check disk usage for:

- **dbspaces**

    Use **onstat** -**d**.

    For more information, see the section on monitoring disk usage in the *Administrator's Guide for Informix Dynamic Server* and the utilities chapter of the *Informix Administrator's Reference*.

- **sbspaces**

    Use **onstat** -**d** or **oncheck** -**cs**,-**cS**, -**ce**, -**pe**, -**ps**, and -**pS**.

    For more information, see the section on monitoring sbspaces in the *Performance Guide for Informix Dynamic Server* and the utilities chapter of the *Informix Administrator's Reference*.

*Tip: When you use **onstat** -**d** to monitor disk usage, the* S *flag in the **Flags** column indicates an sbspace. For each sbspace chunk, the first row displays information about the whole sbspace and user-data area. The second row displays information about the metadata area.*

Increasing the Sizes of Storage Spaces

If you notice that the Enterprise Replication dbspace or sbspace is running out of disk space, you can increase the size of the space by adding chunks to the space.

To add a chunk to a dbspace, use **onspaces** -**a**. For example, to add a 110 kilobyte chunk with an offset of 0 to the **er_dbspace** dbspace, enter:

```
onspaces -a er_dbspace -p /dev/raw_dev2 -o 0 -s 110
```

To add a chunk to an sbspace, use the same **onspaces** command above, however you can specify more information about the chunk that you are adding. After you add a chunk to the sbspace, you must perform a level-0 backup of the root dbspace and the sbspace.

For more information, see the sections on adding chunks to dbspaces and sbspaces in the *Administrator's Guide for Informix Extended Parallel Server* and the utilities chapter of the *Informix Administrator's Reference*.

Recovering when Storage Spaces Fill

When the Enterprise Replication dbspace runs out of disk space, Enterprise Replication raises an alarm and writes a message to the log. When the sbspace runs out of disk space, Enterprise Replication hangs. In either case, you must resolve the problem that is causing Enterprise Replication to spool ("Preventing Memory Queues from Overflowing" on page 8-13) or you must allocate additional disk space ("Increasing the Sizes of Storage Spaces" on page 8-15) before you can continue replication.

Solving Common Configuration Problems

If you experience problems setting up Enterprise Replication, check the following:

- Make sure that you created an sbspace for the row data and set the CDR_QDATA_SBSPACE in the ONCONFIG file.

  For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-11 and "CDR_QDATA_SBSPACE" on page B-9.

- Verify that the trusted environment is set up correctly.

  For more information, see "Setting Up the Trusted Environment" on page 4-5.

- Verify that your SQLHOSTS file is set up properly on each server participating in replication. You must set up database server groups in the SQLHOSTS file.

  For more information, see "Verifying SQLHOSTS" on page 4-5.

- Verify the format of the SQLHOSTS file.

  The network connection (not the shared memory connection) entry should appear immediately after the database server group definition. If the network connection does not appear immediately after the database server group definition, you might see the following error when you run **cdr define server**:

  ```
  command failed -- unable to connect to server specified
  (5)
  ```

  You might also see a message like the following in the message log for the target server:

  ```
  Reason: ASF connect error (-25592)
  ```

- Make sure that the unique identifier for each database server (*i=* in the **options** field of the SQLHOSTS information) is consistent across all nodes in the enterprise.

  For more information, see "Database Server Groups on UNIX" on page 4-6 and Appendix F, "SQLHOSTS Registry Key."

- Verify that the operating system times of the database servers that participate in the replicate are synchronized.

  For more information, see "Time Synchronization" on page 2-17.

- Make sure that the database server has adequate logical log disk space. If the database server does not have enough logical log space at initialization, you will see the following error:

  ```
  command failed -- fatal server error (100)
  ```

- Check the **$INFORMIXDIR/etc/buildsmi.xxx** files to see if a problem occurred when the databases server built the SMI tables.

- Make sure that the databases on all database server instances involved in replication are set to logging (unbuffered logging is recommended).

  For more information, see "Unbuffered Logging" on page 2-10.

- For replicates that use any conflict-resolution rule except *ignore*, make sure that you define *shadow columns* (CRCOLS) for each table involved in replication.

  For more information, see "Preparing Tables for Conflict Resolution" on page 4-21.

- If you defined a participant using SELECT * from *table_name*, make sure that the tables are identical on all database servers defined for the replicate.

  For more information, see "Defining Participants" on page 5-8 and "Participant Type" on page A-89.

- Verify that each replicated column in a table on the source database server has the same data type as the corresponding column on the target server.

  Enterprise Replication does not support replicating a column with one data type to a column on another database server with a different data type.

  The exception to this rule is cross-replication between simple large objects and smart large objects.

  For more information, see "Enterprise Replication Data Types" on page 2-19.

- Verify that all tables defined in a replicate have one PRIMARY KEY constraint.

  For more information, see "Primary Key Constraint" on page 2-11, the *Informix Guide to Database Design and Implementation* and *Informix Guide to SQL: Syntax*.

Enterprise Replication Event Alarms

The following table describes the alarms that are raised by Enterprise Replication and the events that trigger the alarms. For more information on handling event alarms, see the appendix on event alarms in the *Informix Administrator's Reference*.

Situation	Level	Message
Snoopy receives a bad buffer during a log read	ALRM_EMERGENCY	s_cdr_err_logcurrupt
Snoopy sets DDR blockout ("Preventing Blockout Mode" on page 8-14)	ALRM_ATTENTION	s_ddrblock
Snoopy unsets DDR blockout ("Preventing Blockout Mode" on page 8-14)	ALRM_ATTENTION	s_ddrunblock
Snoopy detects that the replay position has been overwritten (page 8-14)	ALRM_EMERGENCY	s_cdr_replay_trounced
An RQM queue runs out of room to spool ("Recovering when Storage Spaces Fill" on page 8-16)	ALRM_EMERGENCY	s_cdr_que_fullsendq
RQM cannot find the replicate in the global catalog for which it has a transaction	ALRM_ATTENTION	s_cdr_badrepid
Unable to allocate memory when analyzing opaque UDTs in collections/rows	ALRM_INFO	s_cdr_nomem
Datasync received transaction that was aborted in first buffer, so there is nothing to spool to ATS/RIS	ALRM_INFO	not yet assigned s_ message
Grouper was unable to apply an undo (rollback to savepoint) to a transaction	ALRM_ATTENTION	s_cdr_undo_error_tx_skipped

(1 of 2)

Situation	Level	Message
Grouper evaluator is aborting	ALRM_EMERGENCY	s_cdr_grp_abort
Grouper fanout is aborting	ALRM_EMERGENCY	s_cdr_grp_abort
Datasync is aborting	ALRM_ATTENTION	s_cdr_ds_abort

(2 of 2)

Appendixes

Section III

Command-Line Utility Reference

The command-line utility (CLU) lets you configure and control Enterprise Replication from the command line on your UNIX or Windows operating system.

This appendix covers the following topics:

- Command Summary
- Command Syntax
- Interpreting the Command-Line Utility Syntax

Command Summary

The following table shows the commands and the page where the command is documented.

Command	Page
cdr change replicate	A-4
cdr change replicateset	A-6
cdr connect server	A-9
cdr define replicate	A-10
cdr define replicateset	A-17
cdr define server	A-20

(1 of 2)

Command	Page
cdr delete replicate	A-25
cdr delete replicateset	A-27
cdr delete server	A-29
cdr disconnect server	A-33
cdr error	A-35
cdr finderr	A-38
cdr list replicate	A-39
cdr list replicateset	A-43
cdr list server	A-45
cdr modify replicate	A-49
cdr modify replicateset	A-54
cdr modify server	A-56
cdr resume replicate	A-59
cdr resume replicateset	A-61
cdr resume server	A-63
cdr start	A-65
cdr start replicate	A-67
cdr start replicateset	A-69
cdr stop	A-71
cdr stop replicate	A-73
cdr stop replicateset	A-75
cdr suspend replicate	A-77
cdr suspend replicateset	A-79
cdr suspend server	A-81

(2 of 2)

Command Syntax

The next sections show the syntax for all the variations of the **cdr** commands.

cdr change replicate

The **cdr change replicate** command allows you to modify an existing replicate by adding or deleting one or more participants.

Syntax
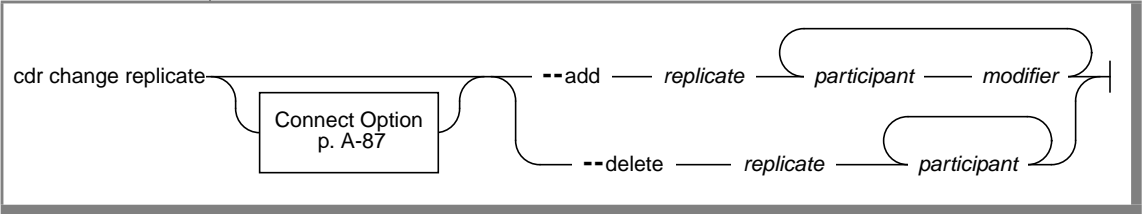
```
cdr change replicate ─┬─────────────────────┬─┬─ --add ─── replicate ─┬─ participant ── modifier ─┬─┤
                      └─ Connect Option     │ │                       └─────────────────────────────┘
                         p. A-87            │ │
                                            │ └─ --delete ─── replicate ─┬─ participant ─┬─
                                            │                            └───────────────┘
```

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate	See "Considerations for the SELECT Statement" on page A-90.	"Participant Modifier" on page A-90
participant	Specifies the database server and table for replication	The participant must exist.	"Participant" on page A-87
replicate	Name of the replicate to change	The replicate must exist.	"Long Identifiers" on page A-86

The following table describes the options to **cdr change replicate**.

Options		
Long Form	**Short Form**	**Meaning**
--**add**	-**a**	Add participants to a replicate.
--**delete**	-**d**	Remove participants from a replicate.

Usage

Use this command to add or delete a participant from a replicate. You can define a replicate that has zero or one participants, but to be useful, a replicate must have at least two participants.

Important: *You cannot start and stop replicates that have no participants.*

Important: *Enterprise Replication adds the participant to the replicate in an inactive state, regardless of the state of the replicate. To activate the new participant, run* ***cdr start replicate*** *with the name of the server group. See "cdr start replicate" on page A-67.*

Examples

The following example adds two participants to the replicate named **repl_1**: **db1@server1:antonio.table** with the modifier `select * from table1`, and **db2@server2:carlo.table2** with the modifier `select * from table2`:

```
cdr change repl -a repl_1 \
    "db1@server1:antonio.table1" "select * from table1" \
    "db2@server2:carlo.table2" "select * from table2"
```

The following example removes the same two participants from replicate **repl_1**:

```
cdr change repl -d repl_1 \
    "db1@server1:antonio.table1" \
    "db2@server2:carlo.table2"
```

See Also

- "cdr define replicate" on page A-10
- "cdr delete replicate" on page A-25
- "cdr list replicate" on page A-39
- "cdr modify replicate" on page A-49
- "cdr resume replicate" on page A-59
- "cdr start replicate" on page A-67
- "cdr stop replicate" on page A-73
- "cdr suspend replicate" on page A-77

cdr change replicateset

The **cdr change replicateset** (or **cdr change replset**) command changes a
replicate set by adding or deleting replicates.

Syntax
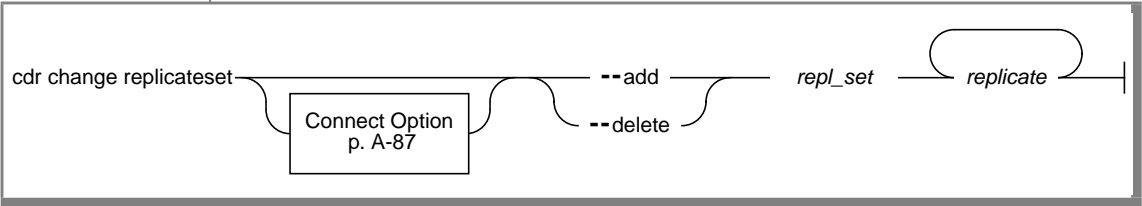
Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set to change	The replicate set must exist.	"Long Identifiers" on page A-86
replicate	Name of the replicates to add to or delete from the set	The replicates must exist.	"Long Identifiers" on page A-86

The following table describes the options to **cdr change replicateset**

Options		
Long Form	**Short Form**	**Meaning**
--**add**	-**a**	Add replicates to a replicate set.
--**delete**	-**d**	Remove replicates from a replicate set.

Usage

Use this command to add replicates to a replicate set or to remove replicates from an exclusive or non-exclusive replicate set:

- If you add a replicate to an exclusive replicate set, Enterprise Replication changes the existing state and replication frequency settings of the replicate to the current properties of the exclusive replicate set.

  If you remove a replicate from an exclusive replicate set, the replicate retains the properties of the replicate set at the time of removal (not the state the replicate was in when it joined the exclusive replicate set).

  When you add or remove a replicate from an exclusive replicate set that is suspended ("Suspending a Replicate Set" on page 7-7) or that is defined with a frequency interval ("Frequency Options" on page A-91), Enterprise Replication transmits all the data in the queue for the replicates in the replicate set up to the point when you added or removed the replicate.

- If you add or remove a replicate to a non-exclusive replicate set, the replicate retains its individual state and replication frequency settings.

Examples

The following example adds the replicates **house** and **barn** to replicate set **building_set**:

```
cdr change replicateset -add building_set house barn
```

The following example removes the replicates **teepee** and **wigwam** from replicate set **favorite_set**:

```
cdr cha replset -del favorite_set teepee wigwam
```

See Also

- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr list replicateset" on page A-43
- "cdr modify replicateset" on page A-54

- "cdr resume replicateset" on page A-61
- "cdr start replicateset" on page A-69
- "cdr stop replicateset" on page A-75
- "cdr suspend replicateset" on page A-79

cdr connect server

The **cdr connect server** command reestablishes a connection to a database server that has been disconnected with a **cdr disconnect server** command.

Syntax

```
cdr connect server ─────────┬─────────────────┬─────── server_group ───────┤
                            │  Connect Option │
                            │     p. A-87     │
                            └─────────────────┘
```
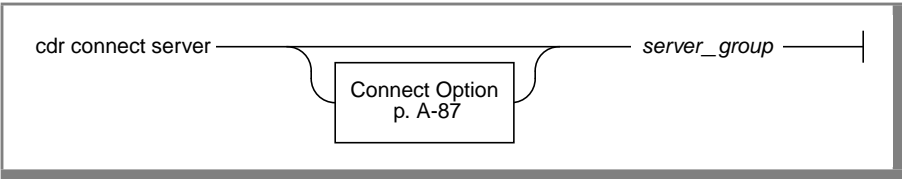
Element	Purpose	Restrictions	Syntax
server_group	Name of database server group to resume	The database server group must be defined for replication and be disconnected.	

Usage

The **cdr connect server** command reestablishes a connection to the server *server_group.*

See Also

- "cdr define server" on page A-20
- "cdr delete server" on page A-29
- "cdr disconnect server" on page A-33
- "cdr list server" on page A-45
- "cdr modify server" on page A-56
- "cdr resume server" on page A-63
- "cdr suspend server" on page A-81

cdr define replicate

The cdr define replicate command defines a replicate in the global catalog.
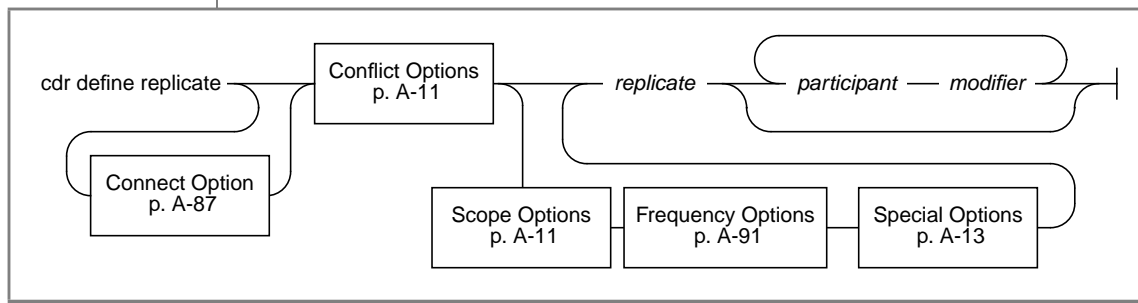
Syntax

```
cdr define replicate ─┬─────────────────┬─ Conflict Options ─┬──────────────────────────────────┬─ replicate ─┬──────────────────────────────────────┬─
                      │                 │    p. A-11          │                                  │             │  ┌── participant ── modifier ──┐    │
                      └─ Connect Option ┘                     │                                  │             └──────────────────────────────────────┘
                          p. A-87                             └─ Scope Options ── Frequency Options ── Special Options ─┘
                                                                  p. A-11        p. A-91            p. A-13
```

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate	See "Considerations for the SELECT Statement" on page A-90.	"Participant Modifier" on page A-90
participant	Name of a participant in the replication	The participant must exist.	"Participant" on page A-87
replicate	Name of the new replicate	The replicate name must be unique.	"Long Identifiers" on page A-86

Usage

To be useful, a replicate must include at least two participants. You can define a replicate that has only one participant, but before you can use that replicate, you must use **cdr change replicate** to add more participants.

Important: You cannot start and stop replicates that have no participants.

When you define a replicate, the replicate does not begin until you explicitly change its state to *active*. For more information, see "cdr start replicate" on page A-67.

> ⬜➡️ **Important:** *Do not create more than one replicate definition for each row and column set of data to replicate. If the participant is the same, Enterprise Replication attempts to insert duplicate values during replication.*

Conflict Options

The **conflict** options specify how Enterprise Replication should resolve conflicts with data arriving at the database server.

```
┌─────────────────┐                    ┌──────────────────────────────────────┐
│ Conflict Options│                    │ Back to cdr define replicate  p. A-10 │
└─────────────────┘                    │ Back to cdr modify replicate  p. A-49 │
                                        └──────────────────────────────────────┘

  ──►── --conflict= ──┬──────────────── ignore ──────────────────────┬──►
                      │                                               │
                      ├── SPL_routine ──┬──────────────┬──────────────┤
                      │                 └── --optimize ─┘              │
                      │                                               │
                      └── timestamp ──┬─────────────────────────────┬─┘
                                      └── , ── SPL_routine ──┬───────┘
                                                             └── --optimize ──┘
```
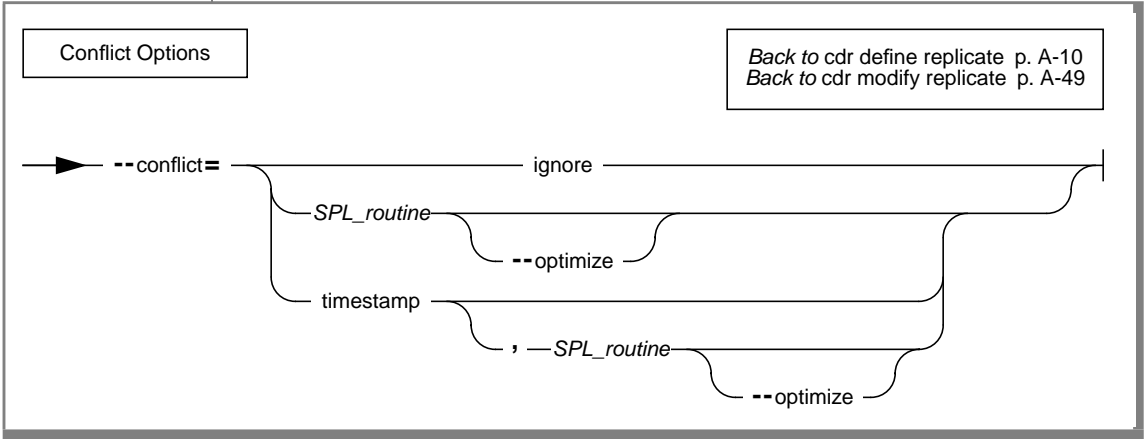
Element	Purpose	Restrictions	Syntax
SPL_routine	SPL routine for conflict resolution	The SPL routine must exist.	"Long Identifiers" on page A-86

Scope Options

The **scope** options specify how Enterprise Replication should handle problems with data arriving at the database server.

```
┌─────────────────┐                    ┌──────────────────────────────────────┐
│ Scope Options   │                    │ Back to cdr define replicate  p. A-10 │
└─────────────────┘                    │ Back to cdr modify replicate  p. A-49 │
                                        └──────────────────────────────────────┘

  ──►── --scope= ──┬──── row ─────┬──►
                   └── transaction ─┘
```
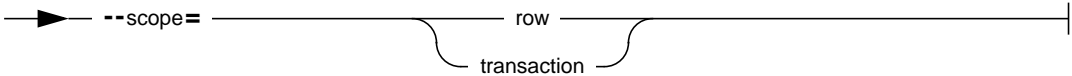
For more information, see "Conflict Resolution" on page 3-11.

The following table describes the **conflict** and **scope** options.

Options		
Long Form	**Short Form**	**Meaning**
--**conflict**=	-**C**	Specifies the rule that will be used for conflict resolution. The action that Enterprise Replication takes depends upon the scope.
--**optimize**	-**O**	Specifies that the SPL routine is *optimized*.
		An optimized SPL routine is called only when a collision is detected and the row to be replicated meets the following two conditions:
		■ It is from the same database server that last updated the local row on the target table.
		■ It has a time stamp greater than or equal to that of the local row.
		When this option is not present, Enterprise Replication always calls the SPL routine defined for the replicate when a conflict is detected.
--**scope**=	-**S**=	Specifies the scope that will be invoked when Enterprise Replication encounters a problem with data or a conflict occurs.
		For more information, see "Scope" on page 3-18.
		If scope is not specified, the default scope is transaction.
		When specifying the scope, you can abbreviate *transaction* to *tra*.

Special Options
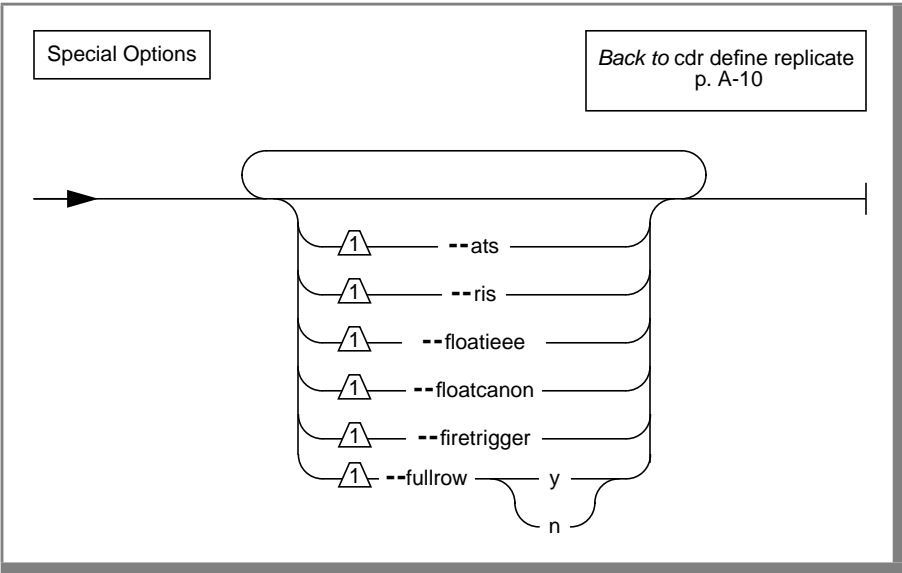
The following table describes the special options to **cdr define replicate**.

Options		
Long Form	**Short Form**	**Meaning**
--**ats**	-**A**	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database. For more information, see "Setting Up Error Logging" on page 5-11 and Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."
--**ris**	-**R**	Activates row-information spooling for replicate row data that fails conflict resolution or encounters replication order problems. For more information, see "Setting Up Error Logging" on page 5-11 and Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."

(1 of 2)

Options		
Long Form	**Short Form**	**Meaning**
--**fullrow y** | **n**	**-f y** | **n**	Specifies to (y) replicate the full row and enable upserts or (n) replicate only changed columns and disable upserts. By default, Enterprise Replication always replicates the entire row and enables upserts. For more information, see "Replicating Only Changed Columns" on page 5-11.
--**floatieee**	**-I**	Transfers replicated floating-point numbers in in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format. Informix recommends that you use this option for all new replicate definitions. For more information, see "Using the IEEE Floating Point or Canonical Format" on page 5-13.
--**floatcanon**	**-F**	Transfers replicated floating-point numbers in machine-independent decimal representation. This format is portable, but can lose accuracy. This format is provided for backward-compatibility only; Informix recommends that you use --**floatieee** for all new replicate definitions. For more information, see "Using the IEEE Floating Point or Canonical Format" on page 5-13.
--**firetrigger**	**-T**	Specifies that the rows that this replicate inserts fire triggers at the destination. For more information, see "Enabling Triggers" on page 5-14.

(2 of 2)

Examples

The following example illustrates the use of **cdr define replicate**:

```
1  cdr define repl --conflict=timestamp,sp1 \
2  --scope=tran --ats --fullrow n --floatieee newrepl \
3  "db1@iowa:antonio.table1" "select * from table1" \
4  "db2@utah:carlo.table2" "select * from table2"
```

Line 1 of the example specifies a primary conflict-resolution rule of *timestamp.*
If the primary rule fails, the SPL routine **sp1** will be invoked to resolve the
conflict. Because no database server is specified here (or on any later line), the
command connects to the database server named in the **INFORMIXSERVER**
environment variable.

Line 2 specifies that the replicate has a transaction scope for conflict-
resolution scope and enables aborted transaction spooling. Enterprise Repli-
cation will only replicate the rows that changed and uses the IEEE floating
point format to send floating-point numbers across dissimilar platforms. The
final item specifies the name of the replicate, **newrepl**.

Line 3 defines the first participant, `"db1@iowa:antonio.table1"`, with the
select statement `"select * from table1"`.

Line 4 defines a second participant, `"db2@utah:carlo.table2"`, with the
select statement `"select * from table2"`.

The next example is the same as the preceding example with the following
exceptions:

Line 1 instructs Enterprise Replication to use the global catalog on database
server **ohio**. Line 2 also specifies that the data should be replicated every five
hours.

```
1  cdr def repl -c ohio -C timestamp,sp1 \
2  -S tran -A -e 5:00 -I newrepl \
3  "db1@iowa:antonio.table1" "select * from table1" \
4  "db2@utah:carlo.table2" "select * from table2"
```

See Also

- "cdr change replicate" on page A-4
- "cdr delete replicate" on page A-25
- "cdr list replicate" on page A-39

- "cdr modify replicate" on page A-49
- "cdr resume replicate" on page A-59
- "cdr start replicate" on page A-67
- "cdr stop replicate" on page A-73
- "cdr suspend replicate" on page A-77

cdr define replicateset

The **cdr define replicateset** (or **cdr define replset**) command defines a replicate set. A replicate set is a collection of several replicates to be managed together.

Important: Enterprise Replication supports replicate sets for Informix Dynamic Server Version 9.3 and later only. You cannot define or modify replicate sets to include replicates with participants that are Version 9.2 and earlier. In addition, replicate sets are different from and are incompatible with replicate groups (in Version 9.2 and earlier).

Syntax

cdr define replicateset ── 1 ── Connect Option p. A-87 ── 1 ── Frequency Options p. A-91 ── --exclusive ── *repl_set* ── *replicate* ──┤
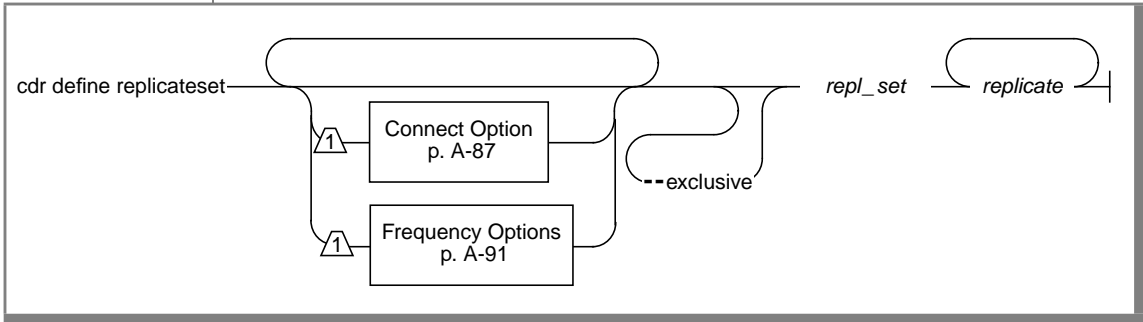
Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to create	The name must be unique and cannot be the same as a replicate name.	"Long Identifiers" on page A-86
replicate	Name of a replicate to be included in the replicate set	The replicate must exist.	"Long Identifiers" on page A-86

The following table describes the option to **cdr define replicateset**.

Options		
Long Form	**Short Form**	**Meaning**
--**exclusive**	-**X**	Creates an exclusive replicate set. For more information, see "Exclusive Replicate Sets" on page 7-4.

Usage

Use the **cdr define replicateset** command to define a replicate set.

Any valid replicate can be defined as part of a replicate set. A replicate can belong to more than one non-exclusive replicate set, but to only one exclusive replicate set. For details on the differences between exclusive and non-exclusive replicate sets, see "Creating Replicate Sets" on page 7-4.

When you create an exclusive replicate set, the state is initially set to active (see "Displaying Information About Replicates" on page A-40).

To create an exclusive replicate set and set the state to inactive

1. Create an empty replicate set.
2. Stop the replicate set.
3. Add replicates to the replicate set.
4. Set the state of the replicate set to active by running **cdr start replicateset**.

Because individual replicates in a non-exclusive replicate set can have different states, the non-exclusive replicate set itself has no state.

Important: Once you create an exclusive replicate set, you cannot change it to non-exclusive, and vice versa.

Example

The following example connects to the default server and defines the non-exclusive replicate set **accounts_set** with replicate participants **repl1**, **repl2**, and **repl3**:

```
cdr def replset accounts_set repl1 repl2 repl3
```

The following example connects to the server **olive** and defines the exclusive replicate set **market_set** with replicate participants **basil** and **thyme**:

```
cdr def replset --connect=olive --exclusive market_set basil thyme
```

See Also

- ■ "cdr change replicateset" on page A-6
- ■ "cdr delete replicateset" on page A-27
- ■ "cdr list replicateset" on page A-43
- ■ "cdr modify replicateset" on page A-54
- ■ "cdr resume replicateset" on page A-61
- ■ "cdr start replicateset" on page A-69
- ■ "cdr stop replicateset" on page A-75
- ■ "cdr suspend replicateset" on page A-79

cdr define server

The **cdr define server** command defines a database server group and all its members (that is, all database servers that are members of the database server group) for Enterprise Replication.

Syntax
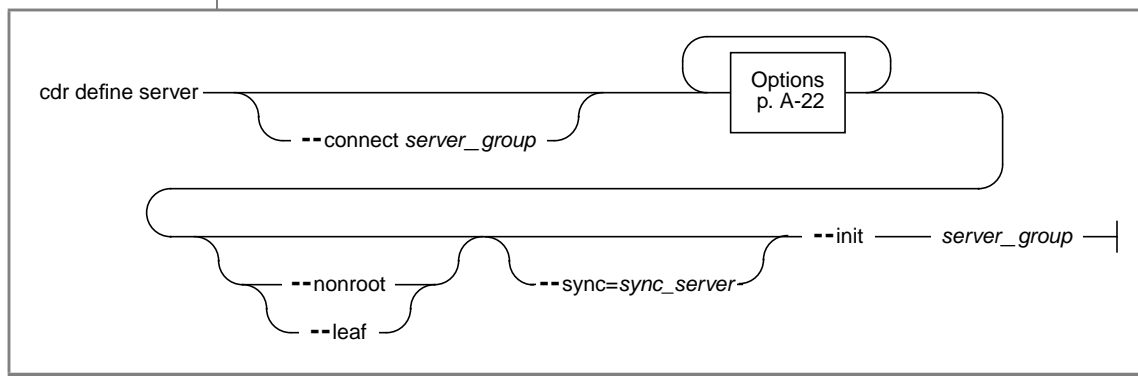
Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to declare for Enterprise Replication	Must be the name of an existing database server group in SQLHOSTS. See "Setting up Database Server Groups" on page 4-5.	
sync_server	Name of server to use for synchronization for all subsequent server definitions to an existing replication system	Must be a server that is registered with Enterprise Replication. The server must be online.	"Long Identifiers" on page A-86

The following table describes the long forms shown in the syntax diagram.

Options		
Long Form	**Short Form**	**Meaning**
--**connect**	**-c**	Connects to the database server that is being defined. If this option is omitted, **$INFORMIXSERVER** must be set to *server_group*.
--**init**	**-I**	Adds *server_group* to the replication system. The *server_group* must be the same as the connection server.
--**leaf**	**-L**	Defines the server as a leaf server. If neither leaf nor nonroot is specified, the server is defined as a root server.
--**nonroot**	**-N**	Defines the server as a nonroot server. If neither leaf nor nonroot is specified, the server is defined as a root server.
--**sync=*sync_server***	**-S=*sync_server***	Uses the global catalog on *sync_server* as the template for the global catalog on the new replication server, *server_group*. Use this option for adding subsequent *server_groups* to an existing replication system. For Hierarchical Routing (HR) topologies, Enterprise Replication also uses the *sync_server* as the new server's parent in the current topology.

Usage

The **cdr define server** command creates the Enterprise Replication global catalog and adds the database server from the *server_group*.

Options

The options allow you to modify the default behavior of **cdr define server**.
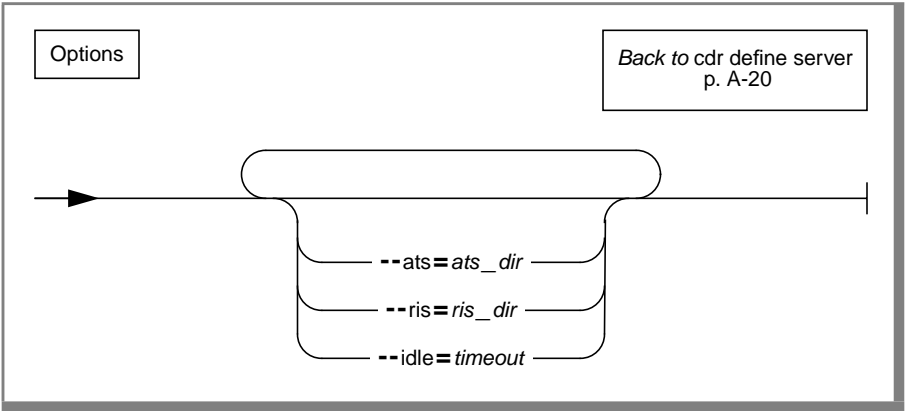
Element	Purpose	Restrictions	Syntax
ats_dir	Name of the directory for Aborted Transaction Spooling	Must be a full pathname. The path for the directory can be no longer than 256 bytes.	Follows naming conventions on your operating system
ris_dir	Name of the directory for Row Information Spooling	Must be a full pathname. The path for the directory can be no longer than 256 characters.	Follows naming conventions on your operating system
timeout	Idle timeout for this replication server	Must be an integer number of minutes. 0 indicates no timeout. The maximum value is 32,767.	Integer

The following table describes the options to **cdr define server.**

Options		
Long Form	**Short Form**	**Meaning**
--**ats**=*ats_dir*	-**A**=*ats_dir*	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database. For more information, see Chapter **8**, "Monitoring and Troubleshooting Enterprise Replication."
--**ris**=*ris_dir*	-**R**=*ris_dir*	Activates row information spooling for replicate row data that fails conflict resolution or encounters replication-order problems. For more information, see Chapter **8**, "Monitoring and Trouble-shooting Enterprise Replication."
--**idle**=*timeout*	-**i**=*timeout*	Causes an inactive connection to be terminated after *timeout* minutes. If time-out is 0, the connection does not time out. The default value is 0.

Examples

The first example defines the first database server in a replication environment. The command connects to the database server **stan**, initializes Enterprise Replication, and sets the idle time-out to 500 minutes. The example also specifies that any files that ATS generates will go into the **/cdr/ats** directory.

```
cdr define server --connect=stan --idle=500 --ats /cdr/ats \
    --init g_stan
```

The following example adds a database server to the replication environment in the first example. The command connects to the database server **oliver**, initializes Enterprise Replication, synchronizes its catalogs with the catalogs on the existing database server **stan**, and defines the database server **oliver** with an idle time-out of 600 minutes. This command also specifies that any files that ATS generates will go into the **/cdr/ats** directory.

```
cdr define server -c oliver -i 600 -A /cdr/ats -I -S \
   g_stan g_oliver
```

See Also

- "cdr connect server" on page A-9
- "cdr delete server" on page A-29
- "cdr disconnect server" on page A-33
- "cdr list server" on page A-45
- "cdr modify server" on page A-56
- "cdr resume server" on page A-63
- "cdr suspend server" on page A-81

cdr delete replicate

The **cdr delete replicate** command deletes a replicate from the global catalog.

Syntax

```
cdr delete replicate ─────┬──────────────────┬────┬─ repl_name ─┬──┤
                          │  ┌────────────┐  │    └─────────────┘
                          └──│Connect Option│──┘
                             │  p. A-87    │
                             └────────────┘
```
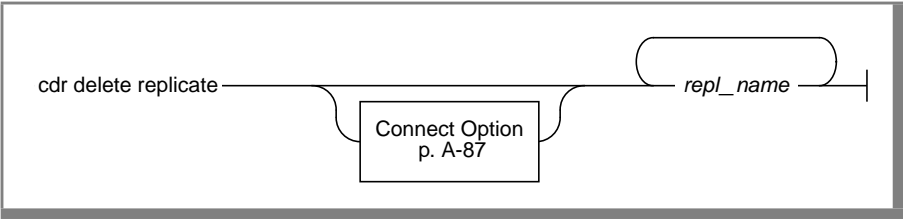
Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate to delete	The replicate must exist.	"Long Identifiers" on page A-86

Usage

The **cdr delete replicate** command deletes the replicate *repl_name* from the global catalog. All replication data for the replicate is purged from the send queue at all participating database servers.

Warning: Avoid deleting a replicate and immediately re-creating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), you might experience failures in the Enterprise Replication system at the time of the operation or later. For more information, see "Operational Considerations" on page 2-8.

Example

The following command connects to the default database server specified by **$INFORMIXSERVER** and deletes the replicate **smile**:

```
cdr del rep smile
```

See Also

- "cdr change replicate" on page A-4
- "cdr define replicate" on page A-10
- "cdr list replicate" on page A-39
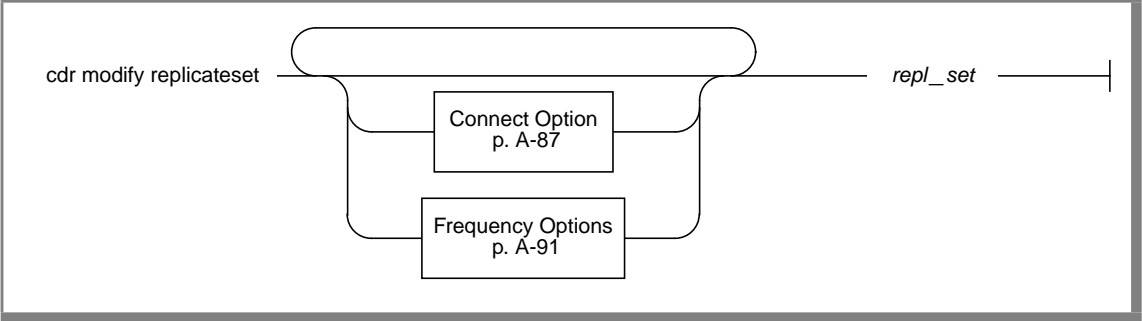- "cdr modify replicate" on page A-49
- "cdr resume replicate" on page A-59
- "cdr start replicate" on page A-67
- "cdr stop replicate" on page A-73
- "cdr suspend replicate" on page A-77

cdr delete replicateset

The **cdr delete replicateset** (or **cdr delete replset**) command deletes a replicate set.

Syntax

```
cdr delete replicateset ──┬──────────────────────┬──┬── repl_set ──┬──┤
                          │   ┌──────────────┐    │  └─────────────┘
                          └──►│ Connect Option│───┘
                              │   p. A-87     │
                              └──────────────┘
```
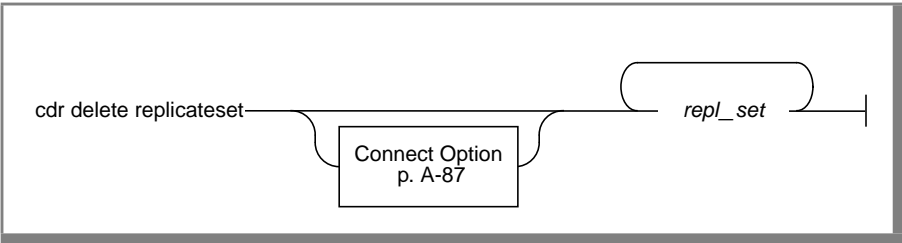
Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to delete	The replicate set must exist.	"Long Identifiers" on page A-86

Usage

The **cdr delete replicateset** command deletes the exclusive or non-exclusive replicate set *repl_set* from the global catalog.

The **cdr delete replicateset** command does not affect the replicates or associated data. When a replicate set is deleted, the individual replicates within the replicate set are unchanged.

Warning: Do not delete time-based exclusive replicate sets. Doing so might result in inconsistent data.

Warning: Avoid deleting a replicate set and immediately re-creating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), you might experience failures in the Enterprise Replication system at the time of the operation or later. For more information, see "Operational Considerations" on page 2-8.

Example

The following example connects to the default database server and deletes the replicate set **accounts_set**:

```
cdr del replset accounts_set
```

See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr list replicateset" on page A-43
- "cdr modify replicateset" on page A-54
- "cdr resume replicateset" on page A-61
- "cdr start replicateset" on page A-69
- "cdr stop replicateset" on page A-75
- "cdr suspend replicateset" on page A-79

cdr delete server

The **cdr delete server** command deletes a database server from the global catalog.

Syntax

```
cdr delete server ─────┬──────────────────────┬───────── server_group ───┤
                       │   Connect Option     │
                       └──    p. A-87      ────┘
```
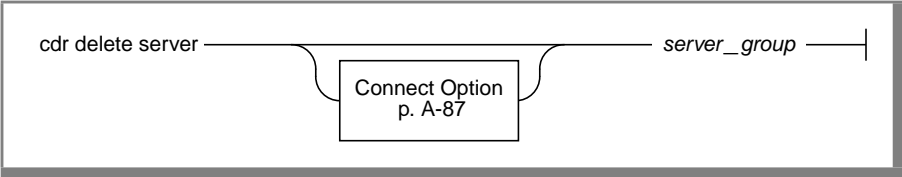
Element	Purpose	Restrictions	Syntax
server_group	Name of database server group to remove from the global catalog	The database server group must be currently defined in Enterprise Replication.	

Usage

The **cdr delete server** command deletes the database server in *server_group* from the global catalog, removes the database server from all participating replicates, and purges all replication data from the send queues for the specified database server. The command shuts down Enterprise Replication on the database server and removes the global catalog from the database server.

When you delete an Enterprise Replication server, you must issue the **cdr delete serve**r command *twice*: once on the server being deleted and once on another server in the enterprise. The first **cdr delete server** removes the Enterprise Replication server from the local global catalog and removes the Enterprise Replication connection to other hosts. The second **cdr delete server** removes the Enterprise Replication server from the other replication servers in the system. For more information, see the "Examples" on page A-30.

You can issue the **cdr delete server** command from any replication server. The only limitation is that you cannot delete a server with children. You must delete the children of a server before deleting the parent server.

You cannot delete a server from the enterprise if it is stopped. To delete a server with **cdr delete server**, you must issue a **cdr start** command first.

Warning: Avoid deleting a replication server and immediately re-creating it with the same name. If you re-create the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), you might experience failures in the Enterprise Replication system at the time of the operation or later. For more information, see "Operational Considerations" on page 2-8.

Examples

This example removes the server **g_italy** from the replication environment. (assume that you issue the commands from the replication server **g_usa**):

```
cdr delete server g_italy
cdr delete server -c italy g_italy
```

The first command performs the following actions:

- Removes **g_italy** from the **usa** global catalog
- Drops the connection from **g_usa** to **g_italy**
- Removes **g_italy** from all participating replicates
- Purges the replication data destined for **g_italy** from send queues
- Broadcasts this delete server command to all other servers (other than **g_italy**) so that they can perform the same actions

The second command connects to server **italy**, and removes Enterprise Replication from **italy**. That is, it removes the **syscdr** database and removes or stops other components of Enterprise Replication.

Figure A-1 shows a replication environment with three replication servers, **g_usa**, **g_italy**, and **g_japan**.
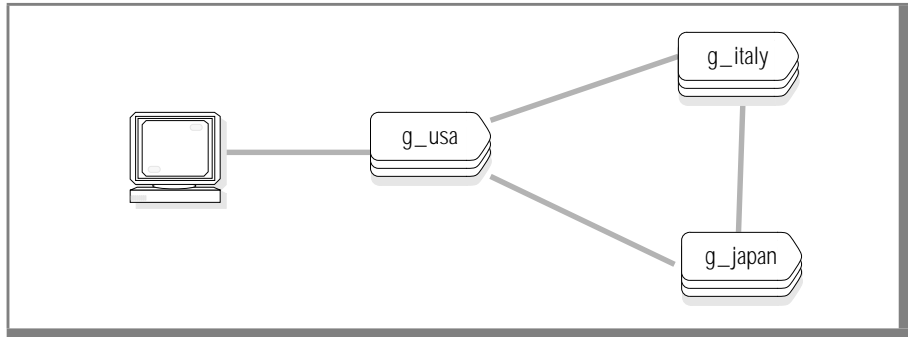
To remove Enterprise Replication from this environment, issue the following commands from the computer where the **usa** replication server resides.

To remove Enterprise Replication from this environment

1. Run:

        cdr delete server g_italy

    This command removes connections between the **italy** replication server and all other servers in the replication system (**usa** and **japan**) and removes any queued data.

2. Run:

        cdr delete server -c italy g_italy

    This command removes all replication information (including the **syscdr** database) from the **italy** database server.

3. Run:

        cdr delete server g_japan
        cdr delete server -c g_japan

    These commands remove the **japan** replication server.

4. Run:

        cdr delete server g_usa

    This command removes the replication information from the **usa** replication server itself.

See Also

- "cdr connect server" on page A-9
- "cdr define server" on page A-20
- "cdr disconnect server" on page A-33
- "cdr list server" on page A-45
- "cdr modify server" on page A-56
- "cdr resume server" on page A-63
- "cdr suspend server" on page A-81

cdr disconnect server

The **cdr disconnect server** command stops a server connection.

Syntax

```
cdr disconnect server ─────┬─────────────────┬───── server_ group ───┤
                           │  Connect Option │
                           │    p. A-87      │
                           └─────────────────┘
```
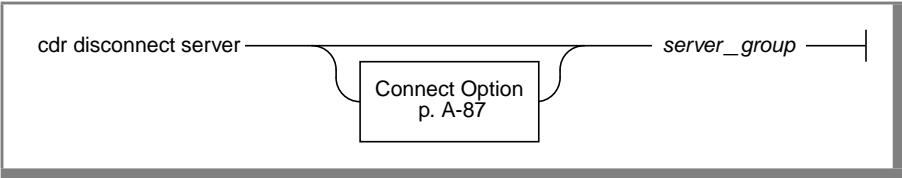
Element	Purpose	Restrictions	Syntax
server_group	Name of the database server group to disconnect	The database server group must be currently active in Enterprise Replication.	

Usage

The **cdr disconnect server** command drops the connection (for example, for a dialup line) between *server_group* and the server specified in the --**connect** option. If the --**connect** option is omitted, the command drops the connection between *server_group* and the default database server (**$INFORMIXSERVER**).

Example

The following example drops the connection between the default database server (**$INFORMIXSERVER**) and the server group **g_store1**:
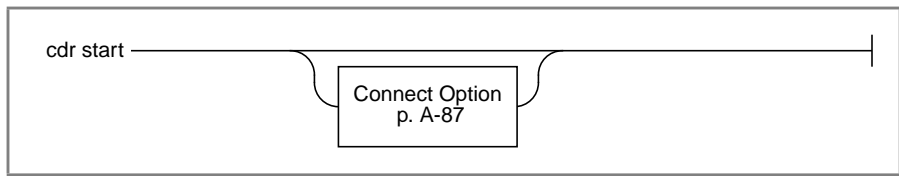
```
cdr disconnect server g_store1
```

See Also

- "cdr connect server" on page A-9
- "cdr define server" on page A-20
- "cdr delete server" on page A-29
- "cdr list server" on page A-45

- "cdr modify server" on page A-56
- "cdr resume server" on page A-63
- "cdr suspend server" on page A-81

cdr error

The **cdr error** command manages the error table and provides convenient displays of errors.

Syntax
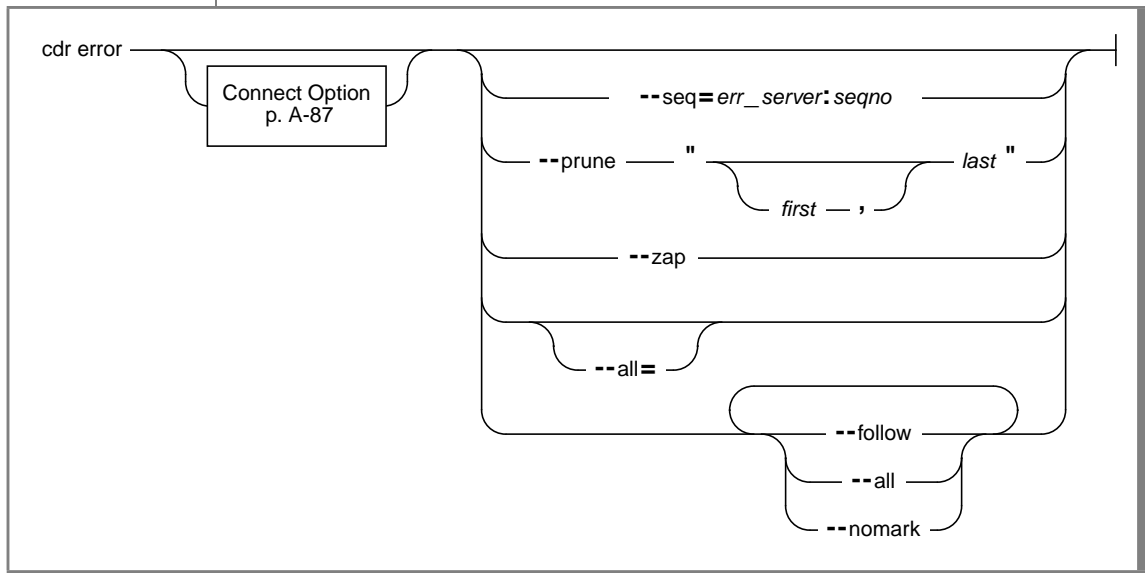
Element	Purpose	Restrictions	Syntax
err_server	Name of database server group that holds the error table	The server must be registered for Enterprise Replication.	"Long Identifiers" on page A-86
first	Start date for a range	You must provide a valid date and time.	"Time of Day" on page A-92
last	Ending date for range	You must provide a later date and time than first.	"Time of Day" on page A-92
seqno	Sequence number of a specific error	You must provide the number of an error in the error table.	Integer

The following table describes the options to **cdr error**:

Options		
Long Form	**Short Form**	**Meaning**
(no options specified)		Print the current list of errors. After the errors have been printed, mark them as *reviewed*. Enterprise Replication does not display errors marked as reviewed.
--**all**	-**a**	When used with the -**move** option, move the error table to *dbspace* on all defined servers. Otherwise, print all errors, including those already reviewed.
--**follow**	-**f**	Continuously monitor the error table.
--**nomark**	-**n**	Do not mark errors as *reviewed*.
--**prune**	-**p**	Prune the error table to those times in the range from *first* to *last*. If *first* is omitted, then all errors earlier than *last* are removed.
--**seq**	-**s**	Remove the (single) error specified by *server:seqno* from the error table.
--**zap**	-**z**	Remove all errors from the error table.

Usage

The **cdr error** command allows you to examine replication errors on any replication server. Sometimes a command succeeds on the server on which it is executed, but fails on one of the remote servers. For example, if you execute **cdr define replicate** on **server1** but the table name is misspelled on **server2**, the command succeeds on **server1** and appears to have completed successfully. You can use **cdr error** -**c server2** to see why replication is failing.

The **cdr error** command also allows you to administer the **cdr error** table remotely. The reviewed flag lets you watch for new errors while keeping the old errors in the table. For example, you could run **cdr error** periodically and append the output to a file.

Examples

The following command displays the current list of errors on database server
hill:

```
cdr error --connect=hill
```

After the errors are displayed, Enterprise Replication marks the errors as
reviewed.

The following command connects to the database server **lake** and removes
from the error table all errors that occurred before the time when the
command was issued:

```
cdr error -c lake --zap
```

The following command deletes all errors from the error table that occurred
at or before 2:56 in the afternoon on May 1, 2000:

```
cdr error -p "2000-05-01 14:56:00"
```

The following command deletes all errors from the error table that occurred
at or after noon on May 1, 2000 and before or at 2:56 in the afternoon on
May 1, 2000:

```
cdr error -p "2000-05-01 14:56:00,2000-05-01 12:00:00"
```

cdr finderr

The **cdr finderr** command looks up a specific Enterprise Replication error number and displays the corresponding error text.

Syntax

cdr finderr ————————————————————————— *ER_error_number* ———|

Element	Purpose	Restrictions	Syntax
error_number	Enterprise Replication error number to look up.	Must be a positive integer.	

You can also view the Enterprise Replication error messages in the file **$INFORMIXDIR/incl/esql/cdrerr.h**.

cdr list replicate

The **cdr list replicate** command displays information about the replicates on the current server.

Syntax

```
cdr list replicate ─────────────────────────────────────────────
                    ┌──────────────┐        ┌─────────────┐
                    │ Connect Option│        │  replicate  │
                    │   p. A-87     │        └─────────────┘
                    └──────────────┘
```
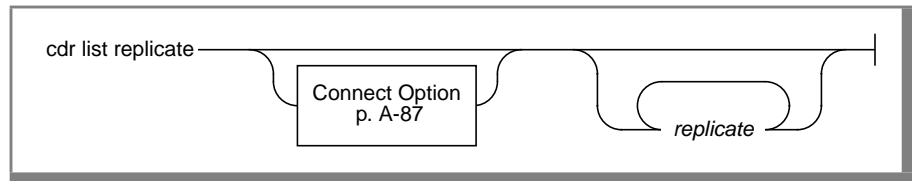
Element	Purpose	Restrictions	Syntax
replicate	Name of the replicates	The replicates must exist.	"Long Identifiers" on page A-86

Usage

The **cdr list replicate** command displays information about replicates. If no replicates are named, the command lists all replicates on the current server. If one or more replicates are named, the command displays detailed information about those replicates.

You do not need to be user **informix** to use this command.

Example

The following example displays a list of the replicates on the current server:

```
cdr list replicate
```

The output from the previous command might be the following:

```
CURRENTLY DEFINED REPLICATES
--------------------------------------------
REPLICATE:    Repl1
STATE:        Inactive
CONFLICT:     Ignore
FREQUENCY:    immediate
QUEUE SIZE:   0
PARTICIPANT:  bank:joe.teller
OPTIONS:      row,ris,ats

REPLICATE:    Repl2
STATE:        Inactive
CONFLICT:     Ignore
FREQUENCY:    immediate
QUEUE SIZE:   0
PARTICIPANT:  bank:joe.account
OPTIONS:      row,ris,ats
```

Displaying Information About Replicates

The STATE field can include the following values.

Value	Description
Active	Specifies that Enterprise Replication captures data from the logical log and transmits it to participants
Definition Failed	Indicates that the replication definition failed on a peer server
Inactive	Specifies that no database changes are captured, transmitted, or processed
Pending	Indicates that a **cdr delete replicate** command has been issued and the replicate is waiting for acknowledgement from the participants
Quiescent	Specifies that no database changes are captured for the replicate or participant
Suspended	Specifies that the replicate captures and accumulates database changes but does not transmit any of the captured data

The CONFLICT field can include the following values.

Value	Description
Ignore	Specifies that the replicate uses the ignore conflict-resolution rule
Timestamp	Specifies that the replicate uses the time stamp conflict-resolution rule
Procedure	Specifies that the replicate uses an SPL routine as the conflict-resolution rule

The FREQUENCY field can include the following values.

Value	Description
immediate	Specifies that replication occurs immediately
every *hh*:*mm*	Specifies that replications occur at intervals (for example, 13:20 specifies every thirteen hours and 20 minutes)
at *day.hh:mm*	Specifies that replications occur at a particular time on a particular day (for example, 15.18:30 specifies on the 15th day of the month at 6:30 P.M.)

The following example specifies the names of replicate:

```
cdr list repl Repl1
```

The following output might result from the previous command:

```
REPLICATE   TABLE                              SELECT
-----------------------------------------------------------
Repl1       bank@g_newyork:joe.teller          select * from joe.teller
Repl1       bank@g_sanfrancisco:joe.teller     select * from joe.teller
Repl1       bank@g_portland:joe.teller         select * from joe.teller
```
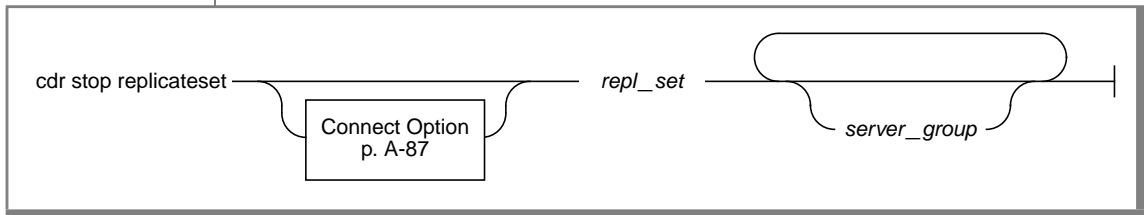
See Also

- "cdr change replicate" on page A-4
- "cdr define replicate" on page A-10
- "cdr delete replicate" on page A-25
- "cdr modify replicate" on page A-49
- "cdr resume replicate" on page A-59
- "cdr start replicate" on page A-67
- "cdr stop replicate" on page A-73
- "cdr suspend replicate" on page A-77

cdr list replicateset

The **cdr list replicateset** (or **cdr list replset**) command displays information about the replication sets defined on the current server.

Syntax

```
cdr list replicateset ──┬──────────────────────┬──┬───────────┬──┤
                        │   ┌──────────────┐    │  │  ┌──────┐ │
                        └──│ Connect Option │──┘  └──│ repl_set │──┘
                            │    p. A-87   │          └──────┘
                            └──────────────┘
```
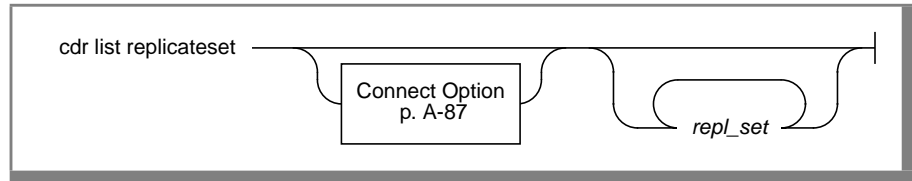
Usage

The **cdr list replicateset** command displays a list of the replicate sets that are currently defined. To list the information about each of the replicates within the replicate set, use **cdr list replicateset** *repl_set*. For more information, see "Displaying Information About Replicates" on page A-40.

You do not need to be user **informix** to use this command.

Examples

The following example displays a list of the replicate sets on the current server:

```
cdr list replicateset
```

The following output might result from the previous command:

```
REPLSET                     PARTICIPANTS
-------------------------------------------------
g1          [Exclusive]     Repl1, Repl4
g2                          Repl2
g3                          Repl3
```

This example displays information for all the replicates in the replicate set **sales_set**:

```
cdr list replset g1
```

The following output might result from the previous command:

```
REPLICATE SET:g1 [Exclusive]
CURRENTLY DEFINED REPLICATES
------------------------------------------------------------------------
REPLICATE:      Repl1
STATE:          Inactive
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    bank:arthur.account
OPTIONS:        row,ris,ats

REPLICATE:      Repl4
STATE:          Inactive
CONFLICT:       Ignore
FREQUENCY:      immediate
QUEUE SIZE:     0
PARTICIPANT:    bank:arthur.teller
OPTIONS:        row,ris,ats
```
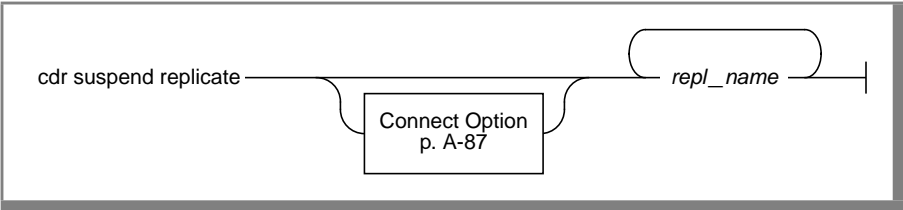
See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr modify replicateset" on page A-54
- "cdr resume replicateset" on page A-61
- "cdr start replicateset" on page A-69
- "cdr stop replicateset" on page A-75
- "cdr suspend replicateset" on page A-79

cdr list server

The **cdr list server** command displays a list of the Enterprise Replication servers that are visible to the current server.

Syntax

```
cdr list server ─────┬─────────────────────┬──────┬──────────────────┬────┤
                     └──┌──────────────┐────┘      └──╭──────────────╮──┘
                        │ Connect Option│              ╰─server_group─╯
                        │    p. A-87    │
                        └──────────────┘
```
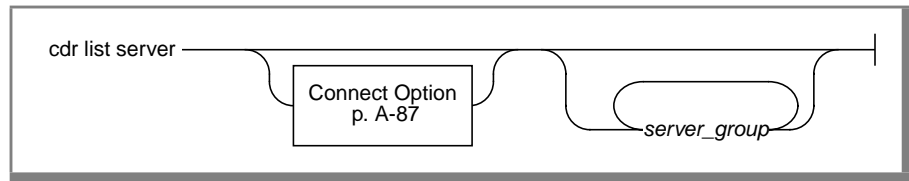
Usage

The **cdr list server** command displays information about servers. You do not need to be user **informix** to use this command.

Listing All Enterprise Replication Servers

When no server-group name is given, the **cdr list server** command lists all database server groups that are visible to the current replication server.

For example, **cdr list server** might give the following output:

```
SERVER         ID STATE     STATUS      QUEUE CONNECTION CHANGED
---------------------------------------------------------------
g_newyork      1 Active     Local       0
g_portland     2 Active     Connected   0     Mar 19 13:48:44
g_sanfrancisco 3 Active     Connected   0     Mar 19 13:48:40
```

The SERVER and ID Columns

The **SERVER** and **ID** columns display the name and unique identifier of the Enterprise Replication server group.

The STATE Column

The **STATE** column can have the following values:

`Active`	Indicates that the server is active and replicating data
`Deleted`	Indicates that the server has been deleted and that it is not capturing or delivering data and the queues are being drained
`Quiescent`	Indicates that the server is in the process of being defined
`Suspended`	Indicates that delivery of replication data to the server is suspended

The STATUS Column

The **STATUS** column can have the following values:

`Connected`	Indicates that the server connection is up
`Connecting`	Indicates that the server is attempting to connect
`Disconnect`	Indicates that the server connection is down in response to an explicit disconnect
`Dropped`	Indicates that the server connection is down due to a network error because the server is unavailable
`Error`	Indicates that an error has occurred (check the log and contact customer support, if necessary)
`Local`	Identifies that this server is the local server as opposed to a remote server
`Timeout`	Indicates that the connection is down due to an idle timeout

The QUEUE Column

The **QUEUE** column displays the size of the queue for the server group.

The CONNECTION CHANGED Column

The **CONNECTION CHANGED** column displays the most recent time that the status of the server connection was changed.

Displaying Details about a Single Replication Server

When the **cdr list server** command includes the name of a database server group, the command displays the attributes of that database server. For example, **cdr list server g_usa** might give the following output:

```
NAME      ID     ATTRIBUTES
-------------------------------------------------------
g_usa      3     timeout=15,atsdir=/w/ats risdir=/w/ris
```

Examples

In Figure A-2 and in the following examples, **usa** and **italy** are root servers, **denver** and **boston** are nonroot servers, and **miami** is a leaf server. The **usa** server is the parent of **denver** and **boston**, and **boston** is the parent of **miami**.
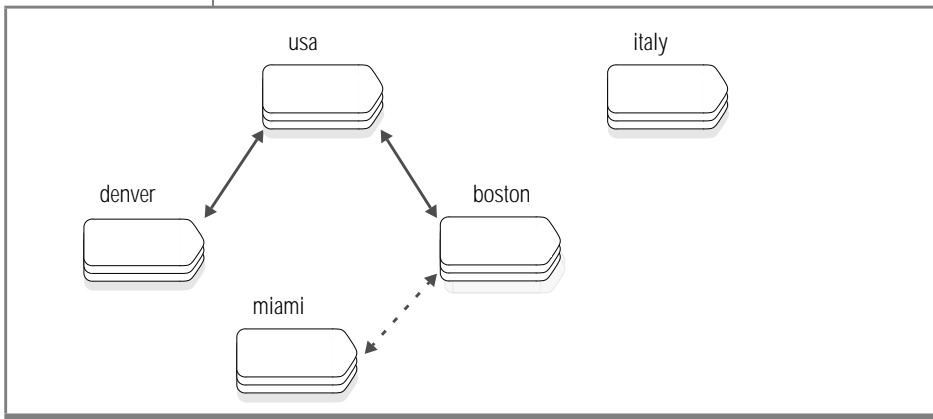
Figure A-2
cdr list server
example

The example is created by running the following commands:

```
cdr list server g_usa
g_usa     1 timeout=15 hub

cdr list server -c denver g_denver
g_denver  27 root=g_usa

cdr list server -c italy g_denver
g_denver  27 root=g_usa forward=g_usa

cdr list server g_miami
g_miami   4 root=g_boston leaf
```

See Also

- ■ "cdr connect server" on page A-9
- ■ "cdr define server" on page A-20
- ■ "cdr delete server" on page A-29
- ■ "cdr disconnect server" on page A-33
- ■ "cdr modify server" on page A-56
- ■ "cdr resume server" on page A-63
- ■ "cdr start" on page A-65
- ■ "cdr suspend server" on page A-81

cdr modify replicate

The **cdr modify replicate** command modifies replicate attributes.

Syntax
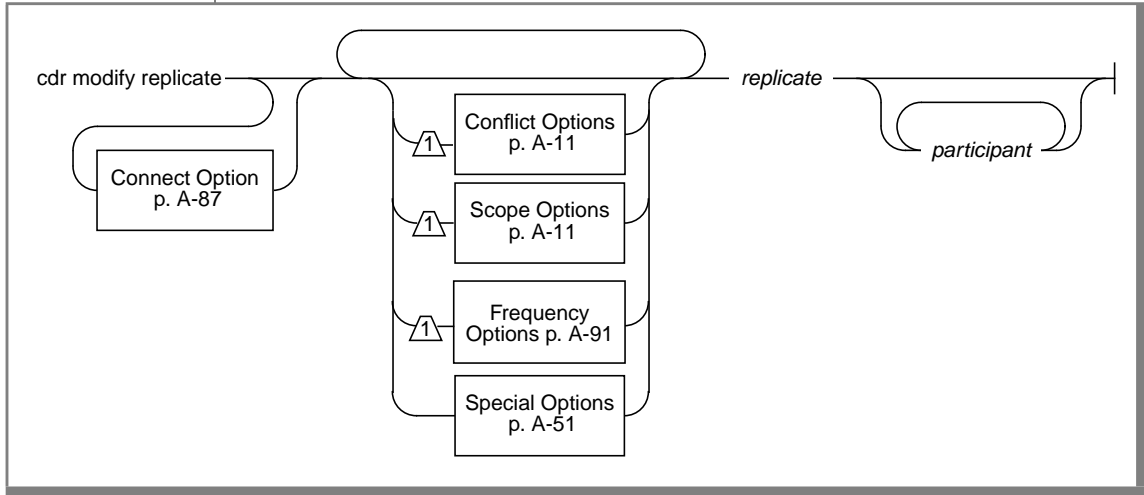
Element	Purpose	Restrictions	Syntax
participant	Name of a participant in the replication	The participant must be a member of the replicate.	"Participant" on page A-87
replicate	Name of the replicate to modify	The replicate name must exist.	"Long Identifiers" on page A-86

Usage

The **cdr modify replicate** command modifies the attributes of a replicate or of one or more participants in the replicate. You can also change the mode of a participant. If the command does not specify participants, the changes apply to all participants in the replicate.

For attribute information, see "cdr define replicate" on page A-10.

To add or delete a participant, see "cdr change replicate" on page A-4.

> **Important:** *If you change the conflict-resolution rule with **cdr modify replicate**, you must also explicitly state SCOPE parameter, even if the SCOPE value does not change.*

Restrictions

The attributes for **cdr modify replicate** are the same as the attributes for **cdr define replicate**, with the following exceptions:

- You cannot change the machine-independent decimal representation (--**floatcanon**) or IEEE floating point (--**floatieee**) formats.

- You cannot change the conflict resolution from *ignore* to a non-*ignore* option (time stamp, SPL routine, or time stamp and SPL routine). You cannot change a non-*ignore* conflict resolution option to *ignore*.

  However, you can change from time stamp resolution to SPL routine resolution or from SPL routine resolution to time stamp.

- The --**ats**, --**ris**, --**firetrigger**, and --**fullrow** options require a yes (y) or no (n) argument.

Special Options

```
┌─────────────────┐                              ┌──────────────────────────┐
│ Special Options │                              │ Back to cdr modify replicate │
└─────────────────┘                              │         p. A-49          │
                                                 └──────────────────────────┘

    ──▶────────────────────────────────────────────────────────────────────┤
              ┌──── --ats ──────┬── y ──┐
              │                 └── n ──┘
              ├──── --ris ──────┬── y ──┐
              │                 └── n ──┘
              ├──── --firetrigger ──┬── y ──┐
              │                     └── n ──┘
              └──── --fullrow ──────┬── y ──┐
                                    └── n ──┘
```
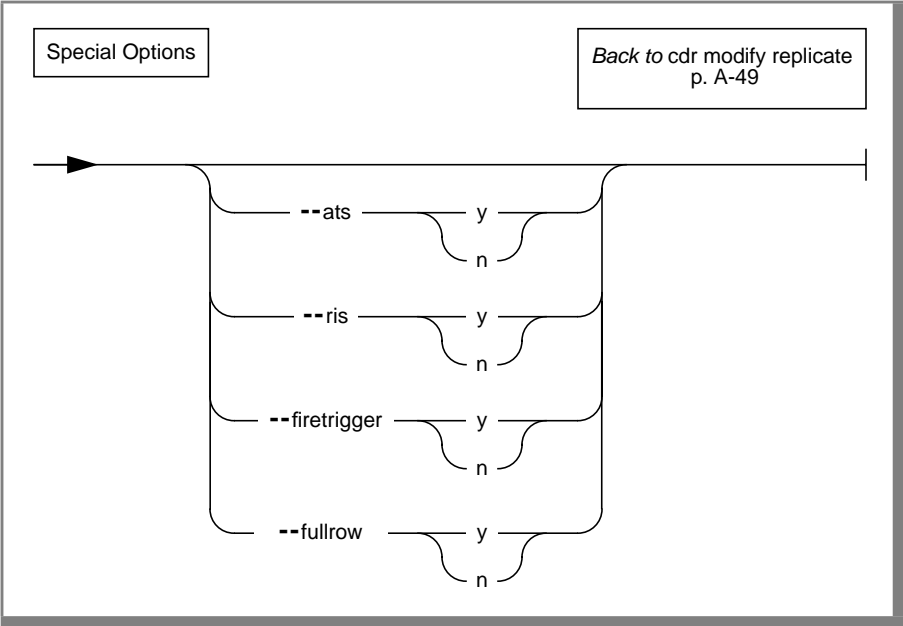
The following table describes the special options to **cdr modify replicate**. For more information on these options, see "Special Options" on page A-13.

Options		
Long Form	**Short Form**	**Meaning**
--**ats y** | **n**	-**A y** | **n**	Activate (y) or deactivate (n) aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
--**ris y** | **n**	-**R y** | **n**	Activate (y) or deactivate (n) row-information spooling for replicate row data that fails conflict resolution or encounters replication-order problems.

(1 of 2)

Options		
Long Form	**Short Form**	**Meaning**
--**firetrigger y** | **n**	-**T y** | **n**	Cause the rows inserted by this replicate to fire (y) or not fire (n) triggers at the destination.
--**fullrow y** | **n**	-**f y** | **n**	Specifies to (y) replicate the full row and enable upserts (default) or (n) replicate only changed columns and disable upserts.

(2 of 2)

Examples

The following example modifies the frequency attributes of replicate **smile** to replicate every five hours:

```
cdr modify repl –every=300 smile
```

The following example modifies the frequency attributes of replicate **smile** to replicate daily at 1:00 A.M.:

```
cdr modify repl -a 01:00 smile
```

The following example modifies the frequency attributes of replicate **smile** to replicate on the last day of every month at 5:00 A.M., to generate ATS files, and not to fire triggers:

```
cdr modify repl -a L.5:00 –A y -T n smile
```

The following example changes the mode of the first participant listed to receive-only and the mode of the second to primary.

```
cdr mod repl smile "R db1@server1:antonio.table1" \
                    "P db2@server2:carlo.table2"
```

See Also

- "cdr change replicate" on page A-4
- "cdr define replicate" on page A-10
- "cdr delete replicate" on page A-25
- "cdr list replicate" on page A-39
- "cdr resume replicate" on page A-59

■ "cdr start replicate" on page A-67

■ "cdr stop replicate" on page A-73

■ "cdr suspend replicate" on page A-77

cdr modify replicateset

The **cdr modify replicateset** (or **cdr modify replset**) command modifies all the replicates in a replicate set.

Syntax

```
cdr modify replicateset ─┬─────────────────────────┬─ repl_set ─┤
                         ├─ Connect Option         │
                         │   p. A-87               │
                         └─ Frequency Options       ┘
                             p. A-91
```

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to modify	The replicate set must exist.	"Long Identifiers" on page A-86

Usage

The **cdr modify replicateset** command modifies the attributes of all the replicates in the replicate set *repl_set*. To add or delete replicates from a replicate set, use the **cdr change replicateset** command ("cdr change replicateset" on page A-6).

Important: Once you create a replicateset, you cannot change it from exclusive to non-exclusive or vice versa.

Example

The following example connects to the default server (**$INFORMIXSERVER**) and modifies the replicate set **sales_set** to process replication data every hour:

```
cdr mod replset --every 60 sales_set
```

See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr list replicateset" on page A-43
- "cdr resume replicateset" on page A-61
- "cdr start replicateset" on page A-69
- "cdr stop replicateset" on page A-75
- "cdr suspend replicateset" on page A-79

cdr modify server

The **cdr modify server** command modifies the Enterprise Replication
attributes of a database server.
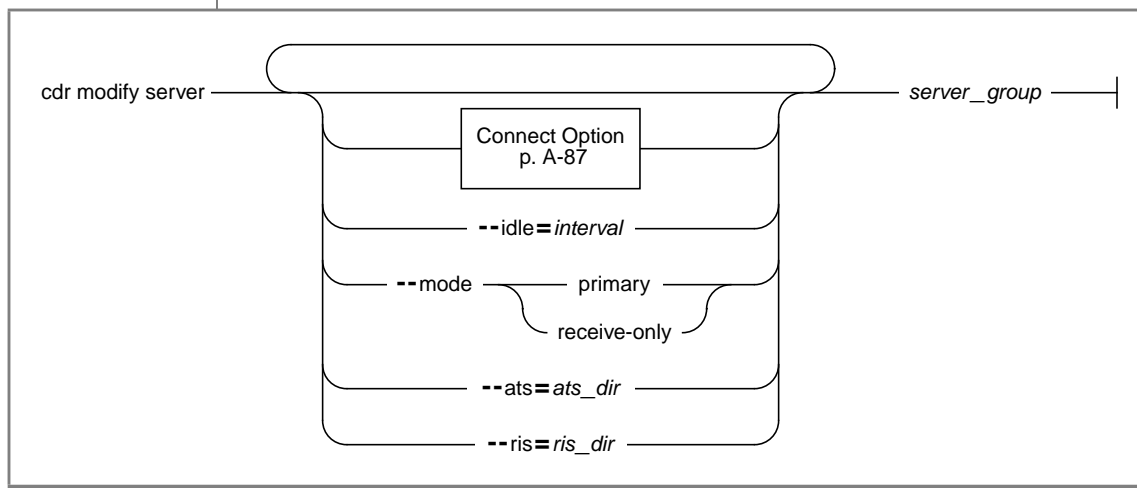
Syntax

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to modify	The database server group must be defined in Enterprise Replication.	
interval	Idle time-out for this server	The value must be an integer >=0. 0 = no timeout.	Integer number of minutes.
ats_dir	Name of Aborted Transaction Spooling directory	The directory must be a full pathname.	Directory name on your operating system.
ris_dir	Name of the Row Information Spooling directory	The directory must be a full pathname.	Directory name on your operating system.

Usage

The **cdr modify server** command modifies the replication server
server_group.

The following table describes the options to **cdr modify server**.

Options		
Long Form	**Short Form**	**Meaning**
--**idle**	**-i**	Causes an inactive connection to be terminated after *idle* time
--**mode**	**-m**	Changes the mode of all replicates using this server to primary (p) or to receive-only (r)
--**ats**	**-A**	Activates aborted-transaction spooling for replicate transactions that fail to be applied to the target database
		For more information, see Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."
--**ris**	**-R**	Activates row-information spooling for replicate-row data that fails conflict resolution or encounters replication-order problems
		For more information, see Chapter 8, "Monitoring and Troubleshooting Enterprise Replication."

Examples

The following example connects to the database server **paris** and modifies
the idle time-out of server group **g_rome** to 10 minutes. ATS files go into the
directory **/cdr/atsdir**.

```
cdr modify server -c paris -i 10 -A /cdr/atsdir g_rome
```

The following example connects to the default database server and sets the
modes of all participants on **g_geometrix** to primary:

```
cdr mod ser -m primary g_geometrix
```

See Also

- "cdr connect server" on page A-9
- "cdr define server" on page A-20
- "cdr delete server" on page A-29
- "cdr disconnect server" on page A-33
- "cdr list server" on page A-45
- "cdr resume server" on page A-63
- "cdr suspend server" on page A-81

cdr resume replicate

The **cdr resume replicate** command resumes delivery of replication data.

Syntax

```
cdr resume replicate ─────┬─────────────────────┬────────┤ repl_name ├────┤
                          └──┤ Connect Option    ├──┘
                             │      p. A-87      │
```
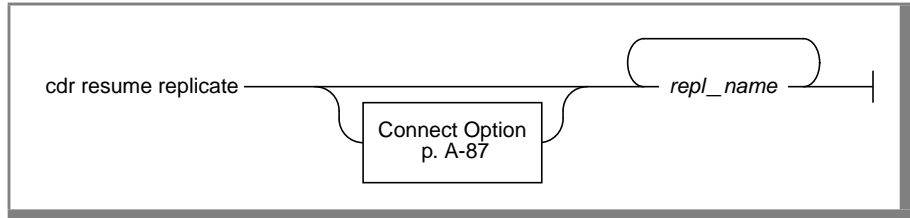
Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate to change to active state.	The replicate must be suspended.	"Long Identifiers" on page A-86

Usage

The **cdr resume replicate** command causes all participants in the replicate *repl_name* to enter the active state.

For more information on replicate states, refer to "The STATE Column" on page A-46.

*Important: If a replicate belongs to an exclusive replicate set ("Exclusive Replicate Sets" on page 7-4), you cannot run **cdr resume replicate** to resume that individual replicate. You must use **cdr resume replicateset** to resume all replicates in the exclusive replicate set.*

If a replicate belongs to a non-exclusive replicate set, you can resume the individual replicates in the set.

Example

The following example connects to the default database server (**$INFORMIXSERVER**) and resumes the replicate **smile**:

```
cdr res repl smile
```

See Also

- "cdr change replicate" on page A-4
- "cdr define replicate" on page A-10
- "cdr delete replicate" on page A-25
- "cdr list replicate" on page A-39
- "cdr modify replicate" on page A-49
- "cdr start replicate" on page A-67
- "cdr stop replicate" on page A-73
- "cdr suspend replicate" on page A-77

cdr resume replicateset

The **cdr resume replicateset** (or **cdr resume replset**) command resumes delivery of replication data for all the replicates in a replicate set.

Syntax

cdr resume replicateset ─── Connect Option p. A-87 ─── *repl_set* ───
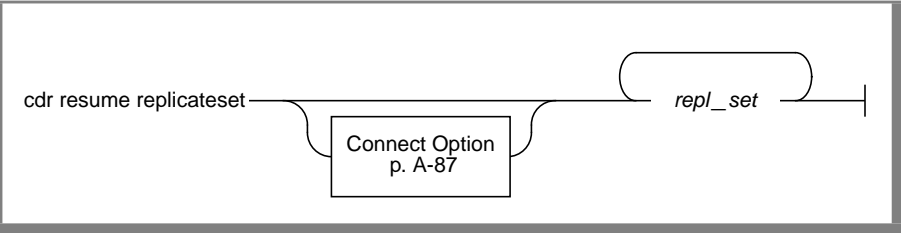
Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to resume	None	"Long Identifiers" on page A-86

Usage

The **cdr resume replicateset** command causes all replicates contained in the replicate set *repl_set* to enter the active state for all participants.

*Important: If not all the replicates in a non-exclusive replicate set are suspended, the **cdr resume replicateset** command displays a warning and only resumes the replicates that are currently suspended.*

For more information on replicate states, refer to "The STATE Column" on page A-46.

Example

The following example connects to the default database server (**$INFORMIXSERVER**) and resumes the replicate set **accounts_set**:

```
cdr res replset accounts_set
```

See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr list replicateset" on page A-43
- "cdr modify replicateset" on page A-54
- "cdr start replicateset" on page A-69
- "cdr stop replicateset" on page A-75
- "cdr suspend replicateset" on page A-79

cdr resume server

The **cdr resume server** command resumes delivery of replication data to a suspended database server.

Syntax

cdr resume server ——[Connect Option p. A-87]—— *to_server_group* ——(*from_server_group*)——

Element	Purpose	Restrictions	Syntax
to_server_group	Name of database server group to which to resume delivery of replication data	The database server group must be currently active in Enterprise Replication.	
from_server_group	Name of the database server group from which to resume sending data to *to_server_group*	The database server group must be currently active in Enterprise Replication.	

Usage

The **cdr resume server** command resumes delivery of replication data to the *to_server_group* database server from the database servers included in the *from_server_group* list. If the *from_server_group* list is omitted, the command resumes replication of data from all database servers participating in the Enterprise Replication system to the *to_server_group*. Replication data must have previously been suspended to the server with the **cdr suspend server** command ("cdr suspend server" on page A-81).

Example

The following example connects to the default server (**$INFORMIXSERVER**) and resumes replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

```
cdr sus serv g_iowa g_ohio g_utah
```

See Also

- "cdr connect server" on page A-9
- "cdr define server" on page A-20
- "cdr delete server" on page A-29
- "cdr disconnect server" on page A-33
- "cdr list server" on page A-45
- "cdr modify server" on page A-56
- "cdr suspend server" on page A-81

cdr start

The **cdr start** command starts Enterprise Replication processing.

Syntax

```
cdr start ─────────────────────────────────────────┐
                    ┌──────────────────┐
                    │  Connect Option  │
                    │     p. A-87      │
                    └──────────────────┘
```

Usage

Use **cdr start** to restart Enterprise Replication after you stop it with **cdr stop**. When you issue **cdr start**, Enterprise Replication brings up all connections to other adjacent replication servers. Replication servers, replicates, and groups that were suspended before the **cdr stop** command was issued remain suspended; no data is sent for the suspended servers, replicates, or groups.

Enterprise Replication resumes evaluation of the logical log (if required for the instance of Enterprise Replication) at the *replay* position. The replay position is the position where Enterprise Replication stops evaluating the logical log when **cdr stop** is executed. If the evaluation process is running and the logical log ID for the replay position no longer exists when Enterprise Replication is started, then the restart partially fails (the database server log contains an error message stating that the replay position is invalid). If the restart partially fails, no database updates performed on the local database server are replicated.

*Warning: Informix recommends that you issue **cdr start** and **cdr stop** with extreme caution.*

Example

The following example restarts Enterprise Replication processing on database server **utah**:

```
cdr sta -c utah
```

See Also

■   "cdr stop" on page A-71

cdr start replicate

The **cdr start replicate** command starts the capture and transmittal of replication transactions.

Syntax

```
cdr start replicate ─────┬──────────────────┬───── repl_name ─────┬──────────────────────────┬─────┤
                         │  Connect Option  │                     │                          │
                         │     p. A-87      │                     └────── server_group ──────┘
                         └──────────────────┘
```
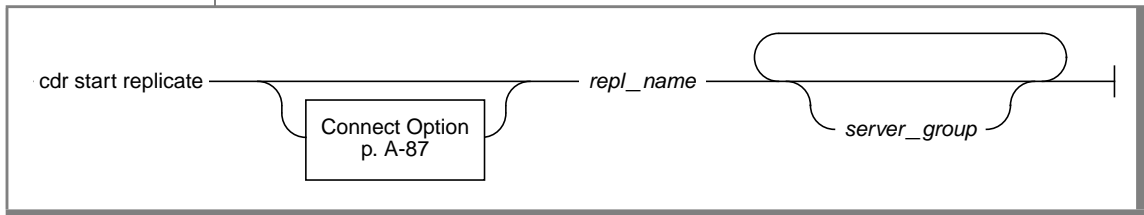
Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate to start	The replicate must exist.	"Long Identifiers" on page A-86
server_group	Name of database server groups on which to start the replicate	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr start replicate** command causes the replicate *repl_name* to enter the active state (capture-send) on the database servers in *server_group.* If no server is specified, the *repl_name* starts on all servers that are included in the replicate. A replicate can have both active and inactive participants. When at least one participant is active, the replicate is active.

Important: You cannot start replicates that have no participants.

*Important: If a replicate belongs to an exclusive replicate set, you cannot run **cdr start replicate** to start that individual replicate. You must use **cdr start replicateset** to start all replicates in the exclusive replicate set.*

Because Enterprise Replication does not process log records that were produced before the **cdr start replicate** command took place, transactions that occur during this period might be partially replicated. To avoid problems, either issue the **cdr start replicate** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLICATION statement until after you successfully start the replicate.

Example

The following command starts the replicate **accounts** on the server groups **g_svr1** and **g_svr2**:

```
cdr sta rep accounts g_svr1 g_svr2
```

See Also

- "cdr change replicate" on page A-4
- "cdr define replicate" on page A-10
- "cdr delete replicate" on page A-25
- "cdr list replicate" on page A-39
- "cdr modify replicate" on page A-49
- "cdr resume replicate" on page A-59
- "cdr stop replicate" on page A-73
- "cdr suspend replicate" on page A-77

cdr start replicateset

The **cdr start replicateset** (or **cdr start replset**) command starts the capture and transmittal of replication transactions for all the replicates in a replicate set.

Syntax

cdr start replicateset ──┬─────────────────────┬── *repl_set* ──┬──────────────┬──────┤
                         └─ Connect Option   ┘   └─ *server_group* ─┘
                            p. A-87
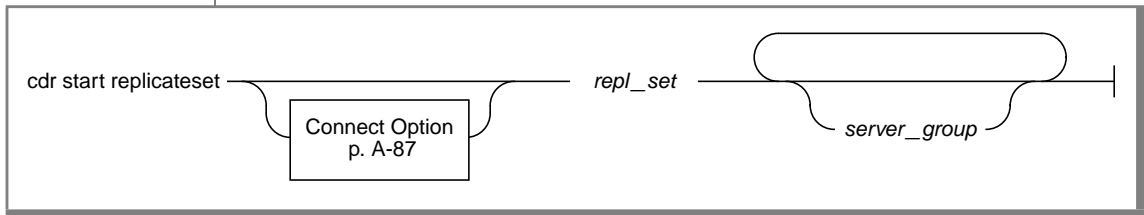
Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to start	The replicate set must exist.	"Long Identifiers" on page A-86
server_group	Names of database server groups on which to start the replicate set	The database server groups must be defined for Enterprise Replication.	

Usage

The replicates defined in the replicate set *repl_set* enter the active state (capture-send) on the database servers in *server_group*.

If the *server_group* list is omitted, the replicate set *repl_set* enters the active state for all database servers participating in the replicate set.

Because Enterprise Replication does not process log records that were produced before the **cdr start replicateset** command took place, transactions that occur during this period might be partially replicated. To avoid problems, either issue the **cdr start replicateset** command on an idle system (no transactions are occurring) or use the BEGIN WORK WITHOUT REPLI-CATION statement until after you successfully start the replicates in the replicate set.

Important: *If not all the replicates in a non-exclusive replicate state are inactive, the* **cdr start replicateset** *command displays a warning and only starts the replicates that are currently inactive.*

Example

The following example connects to the default database server specified by **$INFORMIXSERVER** and starts the replicate set **accounts_set** on the server groups **g_hill** and **g_lake**:

```
cdr sta replset accounts_set g_hill g_lake
```

See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr list replicateset" on page A-43
- "cdr modify replicateset" on page A-54
- "cdr resume replicateset" on page A-61
- "cdr stop replicateset" on page A-75
- "cdr suspend replicateset" on page A-79

cdr stop

The **cdr stop** command stops Enterprise Replication processing.

Syntax

```
cdr stop ─────────────────┬──────────────────────┬──────────────────────┤
                          │  Connect Option      │
                          │  p. A-87             │
                          └──────────────────────┘
```
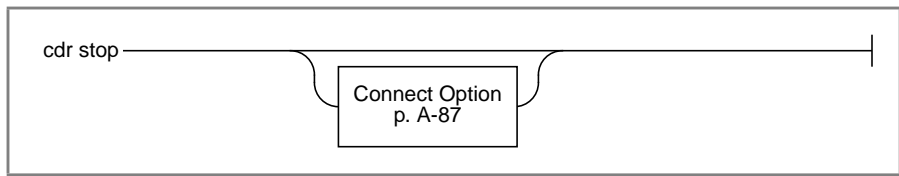
Usage

In most situations, Enterprise Replication starts when **cdr define server** is first executed. The replication threads remain running until the database server is shut down or until the local database server is deleted with the **cdr delete server** command. If you shut down the database server while Enterprise Replication is running, replication begins again when you restart the database server.

Under rare conditions, you might want to temporarily stop the Enterprise Replication processing without stopping the database server. The **cdr stop** command shuts down all Enterprise Replication threads in an orderly manner; however no data to be replicated is captured. When the shutdown of Enterprise Replication is complete, the message CDR shutdown complete appears in the database server log file.

After issuing the **cdr stop** command, replication threads remain stopped (even if the database server is stopped and restarted) until you issue a **cdr start** command.

You cannot delete a server from the enterprise if it is stopped. To delete a server with **cdr delete server**, you must issue a **cdr start** command first.

*Warning: If you issue **cdr stop** and database activity continues, the database server from which the command is issued and the other database servers participating in replicates will become inconsistent. To ensure consistency, verify that no database update activity occurs while Enterprise Replication is stopped.*

Example

The following example stops Enterprise Replication processing on database server **paris**. Processing does not resume until a **cdr start** command restarts it:

```
cdr stop -c paris
```

See Also

■ "cdr start" on page A-65

cdr stop replicate

The **cdr stop replicate** command stops the capture and transmittal of trans-
actions for replication.

Syntax
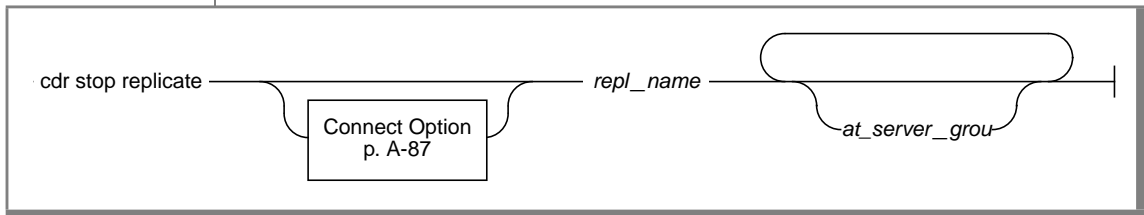
Element	Purpose	Restrictions	Syntax
repl_name	Name of the new replicate	The replicate must be active and not in a replicate group.	"Long Identifiers" on page A-86
at_server_group	List of database server groups on which to stop the replicate	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr stop replicate** command changes the state of the replicate *repl_name*
to inactive (no capture, no send) on the replicate servers in the specified
at_server_group list. In addition, this command deletes any data in the send
queue for the stopped replicate.

Important: You cannot stop replicates that have no participants.

*Warning: If there was any data in the send queue when you issued the **cdr stop
replicate** command, or if any database activity occurs while the replicate is stopped,
the replicate servers will get out of synch. For information on resynchronizing repli-
cation servers, see "Resynchronizing Replication Servers" on page 5-17.*

If you omit the *at_server_group* list, the replicate enters the inactive state on all
database servers participating in the replicate and all send queues for the
replicate are deleted.

> **Important:** If a replicate belongs to an exclusive replicate set, you cannot run **cdr stop replicate** to stop that individual replicate. You must use **cdr stop replicateset** to stop all replicates in the exclusive replicate set.

Example

The following command connects to the database server **lake** and stops the replicate **aRepl** on server groups **g_server1** and **g_server2**:

```
cdr sto rep -c lake aRepl g_server1 g_server2
```

See Also

- "cdr change replicate" on page A-4
- "cdr define replicate" on page A-10
- "cdr delete replicate" on page A-25
- "cdr list replicate" on page A-39
- "cdr modify replicate" on page A-49
- "cdr resume replicate" on page A-59
- "cdr start replicate" on page A-67
- "cdr suspend replicate" on page A-77

cdr stop replicateset

The **cdr stop replicateset** (or **cdr stop replset**) command stops capture and transmittal transactions for all the replicates in a replicate set.

Syntax

```
cdr stop replicateset ─────┬──────────────────┬───── repl_set ──┬─────────────────────┬──┤
                           │  Connect Option  │                 │    server_group     │
                           │     p. A-87      │                 └─────────────────────┘
                           └──────────────────┘
```

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to stop		"Long Identifiers" on page A-86
server_group	Name of database server group on which to stop the replicate group	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr stop replicateset** command causes all replicates in the replicate set *repl_set* to enter the *inactive* state *(*no capture, no send*)* on the database servers in the *server_group* list.

If the *server_group* list is omitted, the replicate set *repl_set* enters the inactive state for all database servers participating in the replicate set.

*Important: If not all the replicates in the non-exclusive replicate set are active, the **cdr stop replicateset** command displays a warning and only stops the replicates that are currently active.*

Example

The following example connects to the database server **paris** and stops the replicate set **accounts_set** on server groups **g_utah** and **g_iowa**:

```
cdr sto replset --connect=paris accounts_set g_utah g_iowa
```

See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr list replicateset" on page A-43
- "cdr modify replicateset" on page A-54
- "cdr resume replicateset" on page A-61
- "cdr start replicateset" on page A-69
- "cdr suspend replicateset" on page A-79

cdr suspend replicate

The **cdr suspend replicate** command suspends delivery of replication data.

Syntax

```
cdr suspend replicate ─────┬───────────────────┬────┬── repl_name ──┬──┤
                           │  ┌─────────────┐   │    └───────────────┘
                           └──│ Connect Option │─┘
                              │  p. A-87       │
                              └─────────────┘
```

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate	The replicate must be active.	"Long Identifiers" on page A-86

Usage

The **cdr suspend replicate** command causes the replicate *repl_name to* enter the suspend state (capture, no send) for all participants.

Warning: When a replicate is suspended, Enterprise Replication holds the replication data in the send queue until the replicate is resumed. If a large amount of data is generated for the replicate while it is suspended, the send queue space can fill, causing data to be lost.

Warning: Enterprise Replication does not synchronize transactions if a replicate is suspended. For example, a transaction that updates tables X and Y will be split if replication for table X is suspended.

*Important: If a replicate belongs to an exclusive replicate set, you cannot run **cdr suspend replicate** to stop that individual replicate. You must use **cdr suspend replicateset** to suspend all replicates in the exclusive replicate set.*

Example

The following example connects to the database server **stan** and suspends the replicate **house**:

```
cdr sus repl --connect stan house
```

See Also

- ■ "cdr change replicate" on page A-4
- ■ "cdr define replicate" on page A-10
- ■ "cdr delete replicate" on page A-25
- ■ "cdr list replicate" on page A-39
- ■ "cdr modify replicate" on page A-49
- ■ "cdr resume replicate" on page A-59
- ■ "cdr start replicate" on page A-67
- ■ "cdr stop replicate" on page A-73

cdr suspend replicateset

The **cdr suspend replicateset** (or **cdr suspend replset**) command suspends delivery of replication data for all the replicates in a replicate set.

Syntax

```
cdr suspend replicateset ─────┬───────────────────┬──────┌──────┐── repl_set ──────────┤
                              └── Connect Option ──┘      └──────┘
                                     p. A-87
```
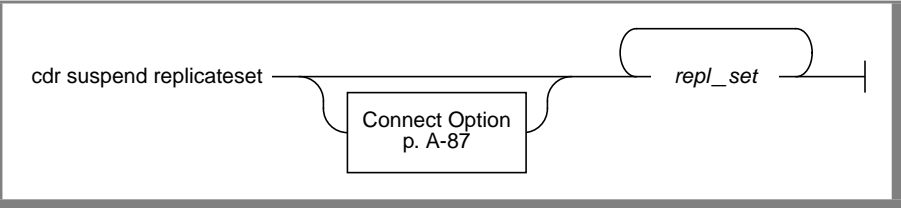
Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to suspend		"Long Identifiers" on page A-86

Usage

The **cdr suspend replicateset** command causes all the replicates in the replicate set *repl_set* to enter the suspend state. Information is captured, but no data is sent for any replicate in the set. The data is queued to be sent when the set is resumed.

Warning: When a replicate set is suspended, Enterprise Replication holds the replication data in the send queue until the set is resumed. If a large amount of data is generated for the replicates in the set while it is suspended, the send queue space can fill, causing data to be lost.

Warning: Enterprise Replication does not synchronize transactions if a replicate in a replicate set is suspended. For example, a transaction that updates tables X and Y will be split if replication for table X is suspended.

*Important: If not all the replicates in the non-exclusive replicate set are active, the **cdr suspend replicateset** command displays a warning and only suspends the replicates that are currently active.*

Example

The following example connects to the default database server specified by
$INFORMIXSERVER and suspends the replicate set **accounts_set**:

```
cdr sus replset account_set
```

See Also

- "cdr change replicateset" on page A-6
- "cdr define replicate" on page A-10
- "cdr delete replicateset" on page A-27
- "cdr list replicateset" on page A-43
- "cdr modify replicateset" on page A-54
- "cdr resume replicateset" on page A-61
- "cdr start replicateset" on page A-69
- "cdr stop replicateset" on page A-75

cdr suspend server

The **cdr suspend server** command suspends the delivery of replication data to a database server from either a specified list of database servers or from all database servers in the enterprise.

Syntax
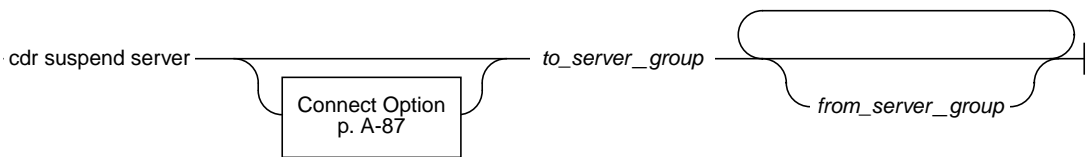
```
cdr suspend server ──┬─────────────────────┬── to_server_group ──┬───────────────────────┬──┤
                     │   ┌───────────────┐ │                     └── from_server_group ──┘
                     └──│ Connect Option │─┘
                         │    p. A-87    │
                         └───────────────┘
```

Element	Purpose	Restrictions	Syntax
to_server_group	Name of database server group to which to suspend delivery of replication data	The database server group must be currently active in Enterprise Replication.	
from_server_group	Name of the database server group from which to stop sending data to *to_server_group*	The database server group must be currently active in Enterprise Replication.	

Usage

The **cdr suspend server** command suspends delivery of replication data to the *to_server_group* database server from the database servers included in the *from_server_group* list. If the *from_server_group* list is omitted, the command suspends replication of data from all database servers participating in the Enterprise Replication system to the *to_server_group*.

The connection to the suspended server is unaffected, and control and acknowledge messages continue to be sent to that server. Enterprise Replication continues to replicate data between all database servers unaffected by the **cdr suspend server** command.

Example

The following example connects to the default server (**$INFORMIXSERVER**) and suspends replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

```
cdr sus serv g_iowa g_ohio g_utah
```

See Also

- ■ "cdr connect server" on page A-9
- ■ "cdr define server" on page A-20
- ■ "cdr delete server" on page A-29
- ■ "cdr disconnect server" on page A-33
- ■ "cdr list server" on page A-45
- ■ "cdr modify server" on page A-56
- ■ "cdr resume server" on page A-63

Interpreting the Command-Line Utility Syntax

This section defines the terminology and conventions used in the descriptions of the command-line utility (CLU).

Each command follows the same approximate format, with the following components:

- Command and its variation

  The command specifies the action that should be taken.

- Options

  The options modify the action of the command. Each option starts with a minus (-) or a double-minus (--).

- Target

  The target specifies the Enterprise Replication object that should be acted upon.

- Other objects

  Other objects specify objects that are affected by the change to the target.

If you enter an incorrect **cdr** command at the command-line prompt, the database server returns a usage message that summarizes the **cdr** commands. For a more detailed usage message, enter **cdr** *variation* -**h**. For example, **cdr list server** -**h**.

*Important: You must be the Enterprise Replication server administrator ("Enterprise Replication Server Administrator" on page 2-4) to execute any of the CLU commands except the **cdr list** options.*

This section covers the following topics:

- Command Abbreviations
- Option Abbreviations
- Option Order
- Long Command-Line Examples
- Long Identifiers
- Connect Option

- Participant
- Return Codes
- Frequency Options

Command Abbreviations

For most commands, each of the words that make up a **cdr** command variation can be abbreviated to three or more characters. For example, the following commands are all equivalent:

```
cdr define replicate
cdr define repl
cdr def rep
```

The exceptions to this rule are the **replicateset** commands, which can be abbreviated to **replset**. For example, the following commands are equivalent:

```
cdr define replicateset
cdr def replset
```

Option Abbreviations

Each option for a command has a long form and a short form. You can use either form, and you can mix long and short forms within a single command. For example, using long forms on UNIX you can write:

UNIX

```
cdr define server --connect=ohio --idle=500 \
   --ats=/cdr/ats --initial utah
```

♦

WIN NT/2000

On Windows, this command line might look like:

```
cdr define server --connect=ohio --idle=500 \
   --ats=D:\cdr\ats --initial utah
```

♦

Using short forms, you can write the previous examples as follows:

UNIX

```
cdr def ser -c ohio -i 500 -A /cdr/ats -I utah
```

♦

WIN NT/2000

```
cdr def ser -c ohio -i 500 -A D:\cdr\ats -I utah
```

♦

The long form is always preceded by a double minus (--). Most (but not all) long forms require an equal sign (=) between the option and its argument. The short form is preceded by a single minus (-) and is the first letter of the long form. The short form never requires an equal sign. However, sometimes the short form is capitalized and sometimes not. To find the correct syntax for the short form, check the table that accompanies each command variation.

Tip: Informix recommends that you use the long forms of options to increase readability.

With the exception of the keyword **transaction**, all keywords (or letter combinations) that modify the command options must be written as shown in the syntax diagrams. For example, in the "Conflict Options" on page A-11, the option (**conflict**) can be abbreviated, but the keyword *ignore* cannot be abbreviated. Both of the following forms are correct:

```
--conflict=ignore
-C ignore
```

Option Order

You can specify the options of the CLU commands in any order. Some of the syntax diagrams in this chapter show the options in a specific order because it makes the diagram easier to read.

Do not repeat any options. The following fragment is incorrect because -**c** appears twice. In most cases, the CLU will catch this inconsistency and flag it as an error. However, if no error is given, the database server uses the last instance of the option. In the following example, the database server uses the value -c=utah:

```
-c=ohio -i=500 -c=utah
```

Tip: Informix suggests that, for ease of maintenance, you always use the same order for your options.

Long Command-Line Examples

The examples in this guide use the convention of a backslash (\) to indicate that a long command line continues on the next line. The following two commands are equivalent. The first command is too long to fit on a single line, so it is continued on the next line. The second example, which uses short forms for the options, fits on one line.

UNIX

```
cdr define server --connect=katmandu --idle=500 \
   --ats=/cdrfiles/ats

cdr def ser -c=katmandu -i=500 -A=/cdrfiles/ats
```

♦

WIN NT/2000

On Windows, these command lines might look like:

```
cdr define server --connect=honolulu --idle=500 \
   --ats=D:\cdrfiles\ats

cdr def ser -c=honolulu -i=500 -A=D:\cdr\ats
```

♦

For information on how to manage long lines at your command prompt, check your operating system documentation.

Long Identifiers

Identifiers are the names of objects that Dynamic Server and Enterprise Replication use, such as database servers, databases, columns, replicates, replicate sets, and so on.

An identifier is a character string that must start with a letter or an underscore. The remaining characters can be letters, numbers, or underscores. On Informix Dynamic Server, all identifiers, including replicates and replicate sets, can be 128 bytes long. However, if you have any database servers in your replication environment that are an earlier version, you must follow the length restrictions for that version.

For more information about identifiers, see the *Informix Guide to SQL: Syntax*.

The location of files, such as the location of the ATS files, can be 256 bytes.

User login IDs can be a maximum of 32 bytes. The owner of a table is derived from a user ID and is thus limited to 32 bytes.

Connect Option

The --**connect** option causes the command to use the global catalog that is on the specified server. If you do not specify this option, the connection defaults to the database server specified by the **INFORMIXSERVER** environment variable.

```
                          ─ -c server ─────────────
───────────►──────┬────────────────────────────────┬──────────┤
                  ├── --connect= server ─────────┤
                  ├── -c server_group ──────────┤
                  └── --connect= server_group ──┘
```
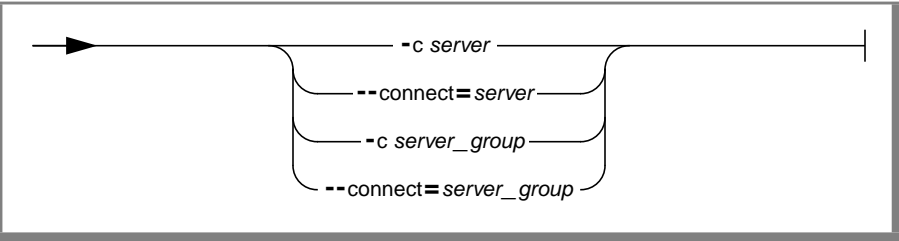
Element	Purpose	Restrictions	Syntax
server	Name of the database server to connect to	The name must be the name of a database server.	"Long Identifiers" on page A-86
server_group	Name of the database server group that includes the database server to connect to	The name must be the name of an existing database server group.	"Long Identifiers" on page A-86

You must use the --**connect** option when you add a database server to your replication environment with the **cdr define server** command.

You might use the --**connect** option if the database server to which you would normally attach is unavailable.

For more information about the global catalog, refer to "Global Catalog" on page 2-6.

Participant

A *participant* defines the data (database, table, and columns) on a database server to be replicated. A replicate contains two or more participants. The participant definition includes the following information:

- Database server group ("Setting up Database Server Groups" on page 4-5)
- Database in which the table resides
- Table name
- Table owner ("Table Owner" on page A-89)
- Participant type ("Participant Type" on page A-89)
- SELECT statement ("Participant Modifier" on page A-90)

Important: *You cannot start and stop replicates that have no participants.*

You must include the server group, database, table owner, and table name when defining a participant, and enclose the entire participant definition in quote marks.

Element	Purpose	Restrictions	Syntax
database	Name of the database that includes the table to be replicated	The database server must be registered with Enterprise Replication.	"Long Identifiers" on page A-86
owner	User ID of the owner of the table to be replicated		"Long Identifiers" on page A-86
server_group	Name of the database server group that includes the server to connect to	The database server group name must be the name of an existing Enterprise Replication server group in SQLHOSTS.	"Long Identifiers" on page A-86
table	Name of the table to be replicated	The table must be an actual table. It cannot be a synonym or a view.	"Long Identifiers" on page A-86

Important: *Do not create more than one replicate definition for each row and column set of data to replicate. If the participant overlaps, Enterprise Replication attempts to insert duplicate values during replication.*

You can define participants with the following commands:

- **cdr define replicate**
- **cdr modify replicate**
- **cdr change replicate**

Table Owner

The **O** (owner) option causes permission checks for *owner* to be applied to the operation (such as insert or update) that is to be replicated and to all actions fired by any triggers. When the **O** option is omitted, all operations are performed with the privileges of user **informix**.

UNIX

If a trigger requires any system-level commands (as specified using the **system( )** command in an SPL statement), the system-level commands are executed as the table owner, if the participant includes the **O** option. ♦

WIN NT/2000

If a trigger requires any system-level commands, the system-level commands are executed as a less privileged user because, on Windows, you cannot impersonate another user without having the password, whether or not the participant includes the **O** option. ♦

Use the **I** option to disable the table-owner option.

Participant Type

In a primary-target replicate, specify the participant type using the **P** (primary) and **R** (receive-only or target) options in the participant definition.

- A primary participant both sends and receives replicate data.
- A target participant only receives data from primary participants.

The replicate can contain multiple primary participants.

In an update-anywhere replicate, do not specify either **P** or **R** for the participant. Enterprise Replication defines the participant as primary in an update-anywhere replication system.

For example, in the following definition, replicate **Rone** is update-anywhere:

```
cdr define repl-c serv1 -C timestamp -S tran Rone \
"db@serv1:owner.table" "select * from table" \
"db@serv2:owner.table" "select * from table"
```

In replicate **Rtwo**, **serv2** is the primary and **serv1** is receive-only.

```
cdr define repl-c serv1 -C ignore -S tran Rtwo \
"R db@serv1:owner.table" "select * from table" \
"P db@serv2:owner.table" "select * from table"
```

For more information, see "Primary-Target Replication System" on page 3-3.

Participant Modifier

The participant *modifier* is a restricted SELECT statement. The participant modifier specifies the rows and columns that will be replicated. The participant modifier must be enclosed in quote marks.

Element	Purpose	Restrictions	Syntax
column	Name of a column in the table specified by the participant	The column must exist.	"Long Identifiers" on page A-86
table	The table specified by the participant	This must be the table name only. You cannot specify an owner or a database server. You cannot specify a synonym or a view.	"Long Identifiers" on page A-86
WHERE Clause	Clause that specifies a subset of table rows to be replicated		WHERE clause syntax; refer to *Informix Guide to SQL: Syntax*

Considerations for the SELECT Statement

The following guidelines apply to a SELECT statement that is used as a participant modifier:

- The statement cannot include a join or a subquery.
- The FROM clause of the SELECT statement can reference only a single table.
- The table in the FROM clause must be the table specified by the participant.
- The columns selected must include the primary key.
- The columns selected can include any columns in the table, including those that contain smart large objects and TEXT and BYTE data.
- The statement cannot perform operations on the selected columns.
- The statement can include an optional WHERE clause.

  The WHERE clause of the SELECT statement of the participant modifier can reference an opaque UDT, as long as the UDT is always stored in row. For more information, see "Considerations for Replicating Opaque Data Types" on page 2-25.

For detailed information about the SELECT statement and WHERE clause, refer to the *Informix Guide to SQL: Syntax*.

Informix recommends that, except for replicating between SERIAL and INT AND between BYTE/TEXT and BLOB/CLOB, you only replicate between like data types. For example, do not replicate between the following:

- CHAR(40) to CHAR(20)
- INT to FLOAT

Return Codes

If the command encounters an error, the database server returns an error message and a return-code value. The message briefly describes the error. For information on interpreting the return code, use the **cdr finderr** command.

Frequency Options

The following commands allow you to specify the interval between replications or the time of day when an action should occur. If you do not specify a time, the action occurs immediately:

- **cdr define replicate**
- **cdr define replicate**
- **cdr modify replicate**
- **cdr modify replicateset**

Frequency Options

```
                                              ──immed──
                    ──every=interval──
                    ──at=time──
```
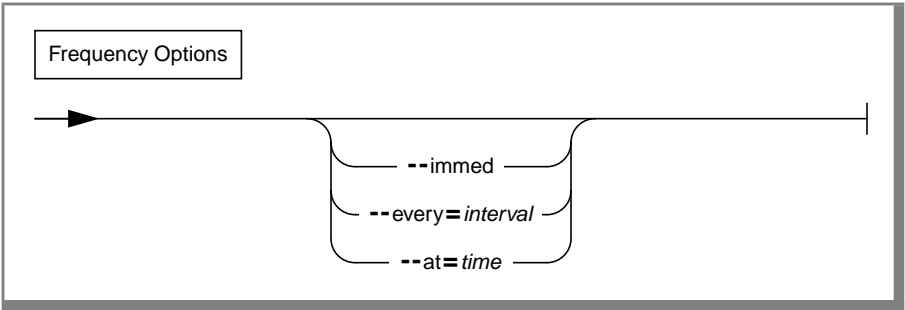
Element	Purpose	Restrictions	Syntax
interval	Time interval for replication	The smallest interval in minutes.	"Intervals" on page A-94
time	Specific time for replication	Time is given with respect to a 24-hour clock.	"Time of Day" on page A-92

The following table describes the frequency options.

Time Options Long Form	Short Form	Meaning
--immed	-i	Action occurs immediately.
--every	-e	Action occurs immediately and repeats at the frequency specified by interval.
--at	-a	Action occurs at the specified day and time.

Time of Day

Enterprise Replication always gives the time of day in 24-hour time. For example, 19:30 is 7:30 P.M. Enterprise Replication always gives times with respect to the local time, unless the **$TZ** environment variable is set. However, Enterprise Replication stores times in the global catalog in Greenwich Mean Time (GMT).

The **-a** *time* option lets you specify the day on which an action should occur. The string *time* can have the following formats:

■   Day of week

To specify a specific day of the week, give either the long or short form of the day, followed by a period and then the time. For example, `--atime Sunday.18:40` or `-a Sun.18:40`, specifies that the action should occur every Sunday at 6:40 P.M.

The day of the week is given in the locale of the client. For example, with a French locale, you might have `--atime Lundi.3:30` or `-a Lun.3:30`. The time and day are in the time zone of the server.

■   Day of month

To specify a specific date in the month, give the date, followed by a period, and then the time. For example, `1.3:00` specifies that the action should occur at 3:00 A.M. on the first day of every month.

The special character `L` represents the last day of the month. For example, `L.17:00` is 5:00 P.M. on the last day of the month.

■   Daily

To specify that replication should occur each day, give only the time. For example, `4:40` specifies that the action should occur every day at 4:40 AM.

Intervals

The -**e** *length* option lets you specify the interval between actions. The *length* of time between replications can be either of the following formats:

■ **The number of minutes**

To specify the number of minutes, specify an integer value. For example, `-e 60` indicates 60 minutes between replications.

Important: If you specify the length of time between replications in minutes, the longest interval allowed is 1966020.

■ **The number of hours and minutes**

To specify hours and minutes, give the number of hours, followed by a colon, and then the number of minutes. For example, `-e 5:45` indicates 5 hours and 45 minutes between replications.

Important: If you specify the length of time in hours and minutes, the longest interval allowed is 32767:59.

Configuration Parameter Reference

The database server configuration file (**$ONCONFIG**) includes the following configuration parameters that affect the behavior of Enterprise Replication:

- CDR_DSLOCKWAIT
- CDR_EVALTHREADS
- CDR_NIFCOMPRESS
- CDR_QUEUEMEM
- CDR_QHDR_DBSPACE
- CDR_QDATA_SBSPACE
- CDR_QDATA_SBFLAGS
- CDR_SERIAL

Important: If you use both DBSERVERNAME and DBSERVERALIASES, DBSERVERNAME should refer to the network connection and not to a shared-memory connection. For information about database server aliases, refer to the Administrator's Guide for Informix Dynamic Server.

CDR_DSLOCKWAIT

default value   5

units   seconds

takes effect   when shared memory is initialized

The CDR_DSLOCKWAIT configuration parameter specifies the number of seconds the Datasync (data synchronization) component waits for database locks to be released. The CDR_DSLOCKWAIT parameter behaves similarly to the SET LOCK MODE statement: while the SET LOCK MODE is set by the end user application, CDR_DSLOCKWAIT is used by Enterprise Replication while applying data at the target database. This parameter is useful in conditions where different sources require locks on the replicated table. These sources could be a replicated transaction from another server or a local application operating on that table.

Transactions that receive updates and deletes from another server in the replicate can abort because of locking problems. If you experience transaction aborts in the Datasync due to lock timeouts like this, you might want to increase the value of this parameter.

CDR_EVALTHREADS

default value	`1,2`
units	evaluator thread instances
range of values	`1` to `129` * the number of CPU VPs for each *value*
takes effect	when shared memory is initialized

Enterprise Replication evaluates the images of a row in parallel to assure high performance. Figure B-1 illustrates how Enterprise Replication uses parallel processing to evaluate transactions for replication.
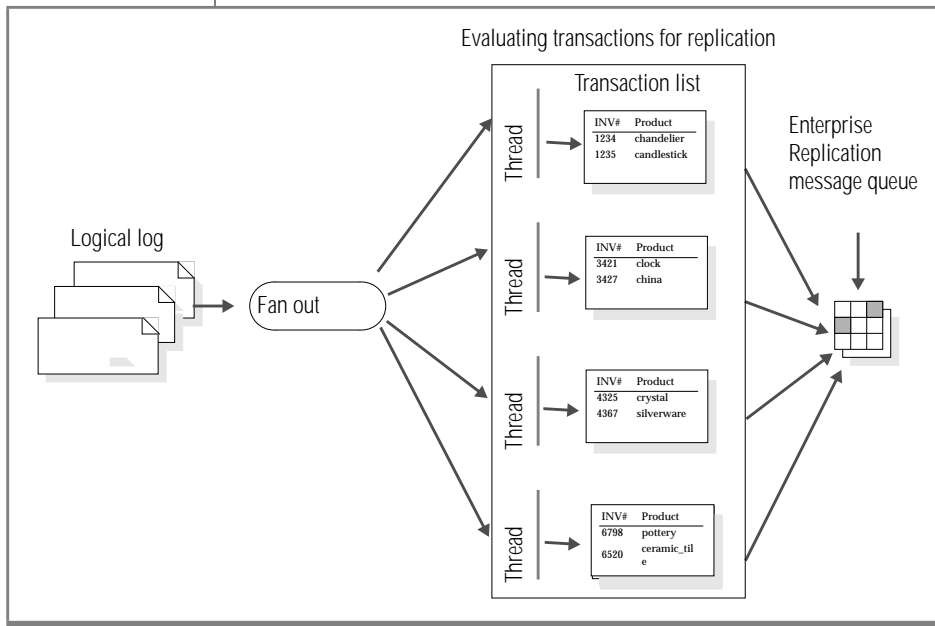
Figure B-1
Processing in Parallel for High Performance

The CDR_EVALTHREADS configuration parameter specifies the number of grouper evaluator threads to create when Enterprise Replication starts and enables parallelism. The format is:

```
(per-cpu-vp,additional)
```

The following table provides four examples of CDR_EVALTHREADS.

Number of Threads	Explanation	Example
1,2	1 evaluator thread per CPU VP, plus 2	For a 3 CPU VP server: (3 * 1) + 2 = 5
2	2 evaluator threads per CPU VP	For a 3 CPU VP server: (3 * 2) = 6
2,0	2 evaluator threads per CPU VP	For a 3 CPU VP server: (3* 2)+0 = 6
0,4	4 evaluator threads for any database server	For a 3 CPU VP server: (3 * 0) +4 = 4

Warning: *Informix recommends that you not configure the total number of evaluator threads to be smaller than the number of CPU VPs in the system.*

CDR_NIFCOMPRESS

default value	0
range of values	■ -1 specifies no compression
	■ 0 specifies to compress only if the target server expects compression
	■ 1 - 9 specifies increasing levels of compression
takes effect	when Enterprise Replication is initialized

The CDR_NIFCOMPRESS (network interface compression) configuration parameter specifies the level of compression that the database server uses before sending data from the source database server to the target database server. Network compression saves network bandwidth over slow links but uses more CPU to compress and decompress the data.

The values have the following meanings.

Value	Meaning
-1	The source database server never compresses the data, regardless of whether or not the target site uses compression.
0	The source database server compresses the data only if the target database server expects compressed data.
1	The database server performs a minimum amount of compression.
9	The database server performs the maximum possible compression.

When Enterprise Replication is defined between two database servers, the CDR_NIFCOMPRESS values of the two servers are compared and changed to the higher compression values.

The compression values determine how much memory can be used to store information while compressing, as follows:

```
   0 = no additional memory
   1 = 128k + 1k = 129k
   2 = 128k + 2k = 130k
...6 = 128k + 32k  = 160k
...8 = 128k + 128k = 256k
   9 = 128k + 256k = 384k
```

Higher levels of CDR_NIFCOMPRESS cause greater compression.

Different sites can have different levels. For example, Figure B-2 shows a set of three root servers connected with LAN and a nonroot server connected over a modem. The CDR_NIFCOMPRESS configuration parameter is set so that connections between A, B, and C use no compression. The connection from C to D uses level 6.
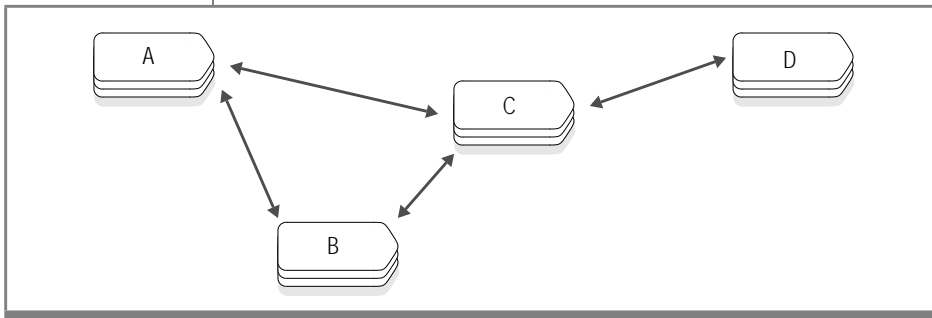
Figure B-2
*Database Servers
with Different
Compression
Levels*

Important: *Informix recommends that you disable NIF compression if the network link performs compression in hardware.*

CDR_QUEUEMEM

default value   4096

units   kilobytes

range of values   > = 500

takes effect   when shared memory is initialized

The CDR_QUEUEMEM configuration parameter specifies the maximum amount of memory that is used for the send and receive queues. If your logical logs are large, the Enterprise Replication reads a large amount of data into queues in memory. You can use CDR_QUEUEMEM to limit the amount of memory devoted to the queues.

The maximum memory used for queued elements on a replication server is the total number of database servers involved in replication, multiplied by the value of CDR_QUEUEMEM.

When you increase the value of CDR_QUEUEMEM, you reduce the number of elements that must be written to disk, which can eliminate I/O overhead. Therefore, if elements are frequently stored on disk, increase the value of CDR_QUEUEMEM. Conversely, if you set the value of CDR_QUEUEMEM too high, you might adversely impact the performance of your system. High values for CDR_QUEUEMEM also increase the time necessary for recovery. Tune the value of CDR_QUEUEMEM for the amount of memory available on your computer.

CDR_QHDR_DBSPACE

default value	the name of the dbspace specified by the ROOTNAME configuration parameter
	For more information, see the *Informix Administrator's Reference*.
takes effect	when Enterprise Replication is initialized

The CDR_QHDR_DBSPACE configuration parameter specifies the location of the dbspace that Enterprise Replication uses to store the transaction record headers spooled from the send and receive queues. By default, Enterprise Replication stores the transaction record headers in the root dbspace. For more information, see "Transaction Record dbspace" on page 4-12.

Warning: *Do not change the value of CDR_QHDR_DBSPACE after you initialize Enterprise Replication.*

CDR_QDATA_SBSPACE

takes effect   when Enterprise Replication is initialized

The CDR_QDATA_SBSPACE configuration parameter specifies the name of the sbspace that Enterprise Replication uses to store spooled transaction row data. Enterprise Replication creates one smart large object per transaction.

Important: *You must set the CDR_QDATA_SBSPACE configuration parameter and create the sbspace specified by CDR_QDATA_SBSPACE before defining a server for replication. If the configuration parameter is not set in ONCONFIG or the sbspace name specified by CDR_QDATA_SBSPACE is invalid, Enterprise Replication fails to define the server.*

For more information, see "Row Data sbspace" on page 4-13 and "Defining Replication Servers" on page 5-5.

Warning: *Do not change the value of CDR_QDATA_SBSPACE after you initialize Enterprise Replication.*

CDR_QDATA_SBFLAGS

default value	0,0
range of values	0 turn on logging
	2 turn off logging
takes effect	each time Enterprise Replication is started

The CDR_QDATA_SBFLAGS configuration parameter specifies the logging flag to use for the smart large object that contains spooled row data (specified by CDR_QDATA_SBSPACE). By default, CDR_QDATA_SBFLAGS is set for no logging; the smart large object specified by CDR_QDATA_SBSPACE is not logged. Informix recommends that you set this parameter to OFF to turn off logging for the smart large object, but that you mirror the sbspace.

For more information, see "LOGGING" on page 4-14.

Tip: Use CDR_QDATA_SBFLAGS at any time to turn logging on or off for the spooled row data smart large object. You must restart Enterprise Replication for the change to take effect.

CDR_SERIAL

default value	`0,0`
units	*delta,offset*
range of values	`0,0` disable control of SERIAL column value generation
	`>0,>0` enable control of SERIAL column value generation
takes effect	when Enterprise Replication is initialized

The CDR_SERIAL configuration parameter enables control over generating values for SERIAL and SERIAL8 columns in tables defined for replication. This is useful for generating unique SERIAL column primary keys in an Enterprise Replication environment.

The format is:

```
CDR_SERIAL delta,offset
```

where:

delta	determines the incremental size of the serial column values
offset	determines the specific number within the delta that will be generated.

Example CDR_SERIAL Value	Resulting Value for the SERIAL Column
`CDR_SERIAL 100,1`	`1`, `101`, `201`, `301`, and so on
`CDR_SERIAL 200,2`	`2`, `202`, `402`, `602`, and so on

You should set the *delta* to greater than the expected number of servers within your enterprise and make sure the *offset* is unique on each server.

When you set CDR_SERIAL, only tables that are marked as the source of a replicate use this method of serial column generation. By default, CDR_SERIAL is set to `0` to disable control over generating SERIAL and SERIAL8 values.

For more information, see "SERIAL Data Types and Primary Keys" on page 2-12.

onstat Command Reference

This appendix describes forms of the **onstat** command that are relevant to Enterprise Replication. The **onstat** utility reads shared-memory structures and provides statistics about the database server that are accurate at the instant that the command executes. The *system-monitoring interface* (SMI) also provides information about the database server. For general information about **onstat** and SMI, refer to the *Administrator's Reference.* For information specific to Enterprise Replication, see Appendix D, "SMI Table Reference."

onstat -g Command Summary

Command	Page
onstat -g ath	C-2
onstat -g cat	C-4
onstat -g ddr	C-4
onstat -g dss	C-5
onstat -g dtc	C-5
onstat -g grp	C-6
onstat -g nif	C-7
onstat -g que	C-8
onstat -g rcv	C-8
onstat -g rep	C-9
onstat -g rqm	C-9

The **onstat** -**g** commands are provided for support and debugging only. You can include only one of these options with each **onstat** -**g** command. This appendix describes the following **onstat** -**g** commands:

onstat -g ath

The **onstat** -**g ath** command prints information about all threads.

The following table summarizes the threads that Enterprise Replication uses. You can use this information about threads when you evaluate memory use. For more information, see the utilities chapter of the *Informix Administrator's Reference.*

Number of Threads	Thread Name	Thread Description
1	ddr_snoopy	Performs physical I/O from logical log, verifies potential replication, and sends applicable log-record entries to Enterprise Replication
1	CDRGfan	Receives log entries and passes entries to evaluator thread
n	CDRGeval*n*	Evaluates log entry to determine if it should be replicated (*n* is the number of evaluator threads specified by CDR_EVALTHREADS) This thread also performs transaction compression on the receipt of COMMIT WORK and queues completed replication messages.
1	CDRCparse	Parses all SQL statements for replicate definitions
1 per connection	CDRNsT*n* CDRNsA*n*	Sending thread for site
1 per connection	CDRNr*n*	Receiving thread for site
2...*n*	CDRACK_*n*	Accepts acknowledgments from site At least 2, up to a maximum of the number of active connections
# CPUs...	CDRD_*n*	Replays transaction on the target system (DataSync thread) At least one thread is created for each CPU virtual processor (VP). The maximum number of threads is 4*(number of CPU VPs).
1	CDRSchedMgr	Schedules internal Enterprise Replication events
0 or 1	CDRM_Monitor	Monitors and adjusts DataSync performance for optimum performance (on the target)
0 or 1	CDRDTCleaner	Deletes (cleans) rows from the deleted rows shadow table when they are no longer needed

onstat -g cat

The **onstat** -**g cat** command prints information from the Enterprise Replication global catalog. The global catalog contains a summary of information about the defined servers, replicates, and replicate sets on each of the servers within the enterprise. For example, use this command to determine the current bitmap mask for a given server, which table matches a given replicate, or whether a server is a root or leaf server.

The **onstat** -**g cat** command has the following formats:

```
onstat -g cat
onstat -g cat scope
onstat -g cat replname
```

The following table describes *replans* and *scope.*

Modifier	Description
replans	The name of a replicate
scope	One of the following values:
	■ servers Print information on servers only
	■ repls Print information on replicates only
	■ full Print expanded information for both replicate servers and replicates

onstat -g ddr

The **onstat** -**g ddr** command prints the status of the Enterprise Replication database log reader. The **ddr**, or **ddr_snoopy**, is an internal component of Enterprise Replication that reads the log buffers and passes information to the grouper. If log reading is blocked, data might not be replicated until the problem is resolved. If the block is not resolved, the database server might overwrite the read (snoopy) position of Enterprise Replication, which means that data will not be replicated. Once this occurs, you must manually resynchronize the source and target databases.

onstat -g dss

The **onstat** -**g dss** command prints detailed statistical information about the activity of individual data sync threads. The data sync thread applies the transaction on the target server. Statistics include the number of applied transactions and failures and when the last transaction from a source was applied.

The **onstat** -**g dss** command has the following formats:

```
onstat -g dss
onstat -g dss modifier
```

The following table describes the values for *modifier*.

Modifier	Action
UDR	Prints summary information about any UDR invocations by the data sync threads.
UDRx	Prints expanded information (including a summary of error information) about any UDR invocations by the data sync threads. The Procid column lists the UDR procedure ID.

onstat -g dtc

The **onstat** -**g dtc** command prints statistics about the delete table cleaner. The delete table cleaner removes rows from the delete table when they are no longer needed.

The -**g dtc** option is used primarily as a debugging tool and by Informix Technical Support.

onstat -g grp

The **onstat** -**g grp** command prints statistics about the grouper. The grouper evaluates the log records, rebuilds the individual log records into the original transaction, packages the transaction, and queues the transaction for transmission.

The -**g grp** option is used primarily as a debugging tool and by Informix Technical Support.

The **onstat** -**g grp** command has the following formats:

```
onstat -g grp
onstat -g grp modifier
```

The following table describes the values for *modifier*.

Modifier	Action
A	Prints all the information printed by the G, T, P, E, R, and S modifiers
E	Prints grouper evaluator statistics
Ex	Prints grouper evaluator statistics, expands user-defined routine (UDR) environments
G	Prints grouper general statistics
L	Prints grouper global list
Lx	Prints grouper global list, expands open transactions
M	Prints grouper compression statistics
Mz	Clears grouper compression statistics
P	Prints grouper table partition statistics
R	Prints grouper replicate statistics
S	Prints grouper serial list head
Sl	Prints grouper serial list
Sx	Prints grouper serial list, expands open transactions

(1 of 2)

T	Prints grouper transaction statistics
UDR	Prints summary information about any UDR invocations by the grouper threads
UDRx	Prints expanded information (including a summary of error information) about any UDR invocations by the grouper threads
	The `Procid` column lists the UDR procedure ID.

(2 of 2)

onstat -g nif

The **onstat** -**g nif** command prints statistics about the network interface, including a summary of the number of buffers sent and received for each site. Because this command reports the status of the various network connections and traffic between the various servers, it can be useful to determine why data is not replicating.

The -**g nif** option is used primarily as a debugging tool and by Informix Technical Support.

The **onstat** -**g nif** command has the following formats:

```
onstat -g nif
onstat -g nif modifier
```

The following table describes the values for *modifier*.

Modifier	Action
all	Prints the sum and the sites
sites	Prints the NIF site context blocks
serverid	Prints information about the replication server whose groupID is serverID
sum	Prints the sum of the number of buffers sent and received for each site

onstat -g que

The **onstat** -**g que** command prints statistics for the high-level queue interface. The queuer manages the logical aspects of the queue. The RQM (reliable queue manager) manages the physical queue.

The -**g que** option is used primarily as a debugging tool and by Informix Technical Support.

onstat -g rcv

The **onstat** -**g rcv** command prints statistics about the receive manager. The receive manager is a set of service routines between the receive queues and data sync.

The **onstat** -**g rcv** command has the following formats:

```
onstat -g rcv
onstat -g rcv serverid
onstat -g rcv full
```

The *serverID* modifier causes the command to print only those output messages received from the replication server whose groupID is *serverid*. The *full* modifier causes the command to print all statistics.

The **onstat** -**g rcv** command includes the Receive Parallelism Statistics section, a summary of the data sync threads by source server.

Field	Description
Server	Source server ID
Tot.Txn.	Total number of transactions applied from this source server
Pending	Number of current transactions in the pending list for this source server
Active	Number of current transactions currently being applied from this source server
MaxPnd	Maximum number of transactions in the pending list queue
MaxAct	Maximum number of transaction in the active list queue

(1 of 2)

AvgPnd	Average depth of the pending list queue
AvgAct	Average depth of the active list queue
CommitRt	Commit rate of transaction from this source server based on transactions per second

(2 of 2)

The -**g rcv** option is used primarily as a debugging tool and by Informix Technical Support.

onstat -g rep

The **onstat** -**g rep** command prints events that are in the queue for the schedule manager. The -**g rep** option is used primarily as a debugging tool and by Informix Technical Support.

The **onstat** -**g rep** command has the following formats:

```
onstat -g rep
onstat -g rep replname
```

The *repl_name* modifier limits the output to those events originated by the replicate named *repl_name*.

onstat -g rqm

The **onstat** -**g rqm** command prints statistics and contents of the low-level queues (send queue, receive queue, ack send queue, sync send queue, and control send queue) managed by the Reliable Queue Manager (RQM). The RQM manages the insertion and removal of items to and from the various queues. The RQM also manages spooling of the in-memory portions of the queue to and from disk. The -**g rqm** option displays the contents of the queue, size of the transactions in the queue, how much of the queue is in memory and on disk, the location of various handles to the queue, and the contents of the various progress tables.

If a queue is empty, no information is printed for that queue.

The **onstat** -**g rqm** command has the following formats:

```
onstat -g rqm
onstat -g rqm modifier
```

The following table describes the values for *modifier*.

Modifier	Action
ACKQ	Prints the ack send queue
CNTRLQ	Prints the control send queue
RECVQ	Prints the receive queue
SENDQ	Prints the send queue
SYNCQ	Prints the sync send queue
FULL	Prints full information about every in-memory transaction for every queue
BRIEF	Prints a brief summary of the number of transactions in each of the queues and the replication servers for which the data is queued
	Use this modifier to quickly identify sites where a problem exists. If large amounts of data are queued for a single server, then that server is probably down or off the network.
VERBOSE	Prints all the buffer headers in memory

When you specify a modifier to select a specific queue, the command prints all the statistics for that queue and information about the first and last in-memory transactions for that queue.

The other modifiers of the **onstat -g rqm** command are used primarily as a debugging tool and by Informix Technical Support.

SMI Table Reference

The system-monitoring interface (SMI) tables in the sysmaster database provide information about the state of the database server. Enterprise Replication uses the SMI tables listed in "SMI Table Summary" to access information about database servers declared to Enterprise Replication.

SMI Table Summary

Table	Description	Reference
syscdrack_buf	Buffers for the acknowledge queue	page D-3
syscdrack_txn	Transactions in the acknowledge queue	page D-3
syscdrctrl_buf	Buffers for the control queue	page D-3
syscdrctrl_txn	Transactions in the control queue	page D-4
syscdrerror	Enterprise Replication error information	page D-4
syscdrpart	Participant information	page D-5
syscdrprog	Contents of the Enterprise Replication progress tables	page D-5
syscdrq	Queued data information (brief)	page D-6
syscdrqueued	Queued data information (detailed)	page D-7

(1 of 2)

Table	Description	Reference
syscdrrecv_buf	Buffers in the receive queue	page D-7
syscdrrecv_txn	Transactions in the receive queue	page D-7
syscdrrepl	Replicate information	page D-8
syscdrreplset	Replicate set information	page D-9
syscdrs	Server information	page D-10
syscdrsend_buf	Buffers in the send queue	page D-11
syscdrsend_txn	Transactions in the send queue	page D-11
syscdrserver	Database server information	page D-12
syscdrsync_buf	Buffers in the synchronization queue	page D-13
syscdrsync_txn	Transactions in the synchronization queue	page D-13
syscdrtx	Transaction information	page D-13

(2 of 2)

SMI Tables for Enterprise Replication

The following sections describe the individual tables of the **sysmaster** database that refer to Enterprise Replication.

syscdrack_buf

The **syscdrack_buf** table contains information about the buffers that form the acknowledgment queue. When the target database server applies transactions, it sends an acknowledgment to the source database server. When the source database server receives the acknowledgment, it can then delete those transactions from its send queue.

For information on the columns of the **syscdrack_buf** table, refer to "Columns of the Buffer Tables" on page D-17.

syscdrack_txn

The **syscdrack_txn** table contains information about the acknowledgment queue. When the target database server applies transactions, it sends an acknowledgment to the source database server. When the source database server receives the acknowledgment, it can then delete those transactions from its send queue.

The acknowledgment queue is an in-memory only queue. That is, it is a volatile queue that is lost if the database server is stopped.

For information on the columns of the **syscdrack_txn** table, refer to "Columns of the Transaction Tables" on page D-16.

syscdrctrl_buf

The **syscdrctrl_buf** table contains buffers that provide information about the control queue. The control queue is a stable queue that contains control messages for the replication system.

For information on the columns of the **syscdrctrl_buf** table, refer to "Columns of the Buffer Tables" on page D-17.

syscdrctrl_txn

The **syscdrctrl_txn** table contains information about the control queue. The control queue is a stable queue that contains control messages for the replication system.

For information on the columns of the **syscdrctrl_txn** table, refer to "Columns of the Transaction Tables" on page D-16.

syscdrerror

The **syscdrerror** table contains information about errors that Enterprise Replication has encountered.

Column	Type	Description
errornum	integer	Error number
errorserv	char(128)	Database server name where error occurred
errorseqnum	integer	Sequence number that can be used to prune single-error table
errortime	datetime year to second	Time error occurred
sendserv	char(128)	Database server name, if applicable, that initiated error behavior
reviewed	char(1)	Y if reviewed and set by DBA N if not reviewed
errorstmnt	text	Error description

syscdrpart

The **syscdrspart** table contains participant information.

Column	Type	Description
replname	lvarchar	Replicate name
servername	char(128)	Database server name
partstate	char(50)	Participant state: ACTIVE, INACTIVE
partmode	char(1)	P = primary database server (read-write) R = target database server (receive only)
dbsname	lvarchar	Database name
owner	lvarchar	Owner name
tabname	lvarchar	Table name

syscdrprog

The **syscdrprog** table lists the contents of the Enterprise Replication progress tables. The progress tables keep track of what data has been sent to which servers and which servers have acknowledged receipt of what data. Enterprise Replication uses the transaction keys and stamps to keep track of this information.

The progress table is two dimensional. For each server to which Enterprise Replication sends data, the progress tables keep progress information on a per-replicate basis.

Column	Type	Description
dest_id	integer	Server ID of the destination server
repl_id	integer	The ID that Enterprise Replication uses to identify the replicate for which this information is valid
source_id	integer	Server ID of the server from which the data originated

(1 of 2)

Column	Type	Description
key_acked_srv	integer	Last key for this replicate that was acknowledged by this destination
key_acked_lgid	integer	Logical log ID
key_acked_lgpos	integer	Logical log position
key_acked_seq	integer	Logical log sequence
tx_stamp_1	integer	Together with tx_stamp2, forms the stamp of the last transaction acknowledged for this replicate by this destination
tx_stamp_2	integer	Together with tx_stamp1, forms the stamp of the last transaction acknowledged for this replicate by this destination

(2 of 2)

syscdrq

The **syscdrq** table contains information about Enterprise Replication queues.

Column	Type	Description
srvid	integer	The identifier number of the database server
repid	integer	The identifier number of the replicate
srcid	integer	The server ID of the source database server. In cases where a particular server is forwarding data to another server, *srvid* is the target and *srcid* is the source that originated the transaction.
srvname	char(128)	The name of the database server
replname	char(128)	Replicate name
srcname	char(128)	The name of the source database server
bytesqueued	integer	Number of bytes queued

syscdrqueued

The **syscdrqueued** table contains data-queued information.

Column	Type	Description
servername	char(128)	Sending to database server name
name	char(128)	Replicate name
bytesqueued	decimal(32,0)	Number of bytes queued for the server servername

syscdrrecv_buf

The **syscdrrecv_buf** table contains buffers that provide information about the data-receive queue. When a replication server receives replicated data from a source database server, it puts this data on the receive queue for processing. On the target side, Enterprise Replication picks up transactions from this queue and applies them on the target.

For information on the columns of the **syscdrrecv_buf** table, refer to "Columns of the Buffer Tables" on page D-17.

syscdrrecv_txn

The **syscdrrecv_txn** table contains information about the data receive queue. The receive queue is an in-memory queue.

When a replication server receives replicated data from a source database server, it puts this data on the receive queue, picks up transactions from this queue, and applies them on the target.

For information on the columns of the **syscdrrecv_txn** table, refer to "Columns of the Transaction Tables" on page D-16.

syscdrrepl

The **syscdrrepl** table contains replicate information.

Column	Type	Description
replname	lvarchar	Replicate name
replstate	char(50)	Replicate state For possible values, refer to "The STATE Column" on page A-46.
freqtype	char(1)	Type of replication frequency: C = continuous I = interval T = time based M = day of month W = day of week
freqmin	smallint	Minute (after the hour) refresh should occur Null if continuous
freqhour	smallint	Hour refresh should occur Null if continuous
freqday	smallint	Day of week or month refresh should occur
scope	char(1)	Replication scope: T = transaction R = row-by-row
invokerow-spool	char(1)	Y = row spooling is invoked N = no row spooling is invoked
invoke transpool	char(1)	Y = transaction spooling is invoked N = no transaction spooling is invoked
primresolution	char(1)	Type of primary conflict resolution: I = ignore T = time stamp S = SPL routine
secresolution	char(1)	Type of secondary conflict resolution: S = SPL routine Null = not configured

(1 of 2)

Column	Type	Description
storedprocname	lvarchar	Name of SPL routine for secondary conflict resolution Null if not defined.
floattype	char(1)	C = converts floating point numbers to canonical format I= converts floating point numbers to IEEE format N= does not convert floating point numbers (sends in native format)
istriggerfire	char(1)	Y = triggers are invoked N = triggers are not invoked
isfullrow	char(1)	Y = sends the full row and enables upserts N = sends only changed columns and disables upserts.

(2 of 2)

syscdrreplset

The **syscdrreplset** table contains replicate set information.

Column	Type	Description
replname	lvarchar	Replicate name
replsetname	lvarchar	Replicate set name

syscdrs

The **syscdrs** table contains information about database servers that have been declared to Enterprise Replication.

Column	Type	Description
servid	integer	Server identifier
servname	char(128)	Database server name
cnnstate	char(1)	Status of connection to this database server: C = Connected D = Connection disconnected (will be retried) T = Idle time-out caused connection to terminate X = Connection closed by user command Connection unavailable until reset by user
cnnstatechg	integer	Time that connection state was last changed
servstate	char(1)	Status of database server: A = Active S = Suspended Q = Quiescent (initial sync state only)
ishub	char(1)	Y = Server is a hub N = Server is not a hub A hub is any replication server that forwards information to another replication server.
isleaf	char(1)	Y = Server is a leaf server N = Server is not a leaf server
rootserverid	integer	The identifier of the root server
forwardnodeid	integer	The identifier of the parent server
timeout	integer	Idle timeout

Although not directly connected, a nonroot server is similar to a root server except it forwards all replicated messages through its parent (root) server. All nonroot servers are known to all root servers and other nonroot servers. A nonroot server can be a terminal point in a tree or it can be the parent for another nonroot server or a leaf server. Nonroot and root servers are aware of all replication servers in the replication environment, including all the leaf servers.

A leaf server is a nonroot server that has a partial catalog. A leaf server has knowledge only of itself and its parent server. It does not contain information about replicates of which it is not a participant. The leaf server must be a terminal point in a replication hierarchy.

syscdrsend_buf

The **syscdrsend_buf** table contains buffers that give information about the send queue. When a user performs transactions on the source database server, Enterprise Replication queues the data on the send queue for delivery to the target servers.

For information on the columns of the **syscdrsend_buf** table, refer to "Columns of the Buffer Tables" on page D-17.

syscdrsend_txn

The **syscdrsend_txn** table contains information about the send queue. When a user performs transactions on the source database server, Enterprise Replication queues the data on the send queue for delivery to the target servers.

For information on the columns of the **syscdrsync_txn** table, refer to "Columns of the Transaction Tables" on page D-16.

syscdrserver

The **syscdrserver** table contains information about database servers declared to Enterprise Replication.

Column	Type	Description
servid	integer	Replication server ID
servername	char(128)	Database server group name
connstate	char(1)	Status of connection to this database server: C = Connected D = Connection disconnected (will be retried) T = Idle time-out caused connection to terminate X = Connection closed by user command Connection unavailable until reset by user
connstate-change	integer	Time that connection state was last changed
servstate	char(50)	Status of this database server: A = Active S = Suspended Q = Quiescent (initial sync state only)
ishub	char(1)	Y = Server is a hub N = Server is not a hub
isleaf	char(1)	Y = Server is a leaf server N = Server connection is not a leaf server
rootserverid	integer	The identifier of the root server
forwardnodeid	integer	The identifier of the parent server
idletimeout	integer	Idle time-out
atsdir	lvarchar	ATS directory spooling name
risdir	lvarchar	RIS directory spooling name

syscdrsync_buf

The **syscdrsync_buf** table contains buffers that give information about the synchronization queue. Enterprise Replication uses this queue only when defining a replication server and synchronizing its global catalog with another replication server.

For information on the columns of the **syscdrsync_buf** table, refer to "Columns of the Buffer Tables" on page D-17.

syscdrsync_txn

The **syscdrsync_txn** table contains information about the synchronization queue. This queue is currently used only when defining a replication server and synchronizing its global catalog with another replication server. The synchronization queue is an in-memory-only queue.

For information on the columns of the **syscdrsync_txn** table, refer to "Columns of the Transaction Tables" on page D-16.

syscdrtx

The **syscdrtx** table contains information about Enterprise Replication transactions.

Column	Type	Description
srvid	integer	Server ID
srvname	char(128)	Name of database server from which data is received
txprocssd	integer	Transaction processed from database server *srvname*
txcmmtd	integer	Transaction committed from database server *srvname*
txabrtd	integer	Transaction aborted from database server *srvname*

(1 of 2)

Column	Type	Description
rowscmmtd	integer	Rows committed from database server *srvname*
rowsabrtd	integer	Rows aborted from database server *srvname*
txbadcnt	integer	Number of transactions with source commit time (on database server *srvname*) greater than target commit time

(2 of 2)

Enterprise Replication Queues

One group of **sysmaster** tables shows information about Enterprise Replication queues. The **sysmaster** database reports the status of these queues in the tables that have the suffixes **_buf** and **_txn**.

The name of each table that describes an Enterprise Replication queue is composed of the following three pieces:

- **syscdr**, which indicates that the table describes Enterprise Replication
- An abbreviation that indicates which queue the table describes
- A suffix, either **_buf** or **_txn**, which specifies whether the table includes buffers or transactions

Selecting from these tables provides information about the contents of each queue. For example, the following SELECT statement returns a list of all transactions queued on the send queue:

```
SELECT * FROM syscdrsend_txn
```

The following example returns a list of all transactions queued on the in-memory send queue and returns the number of buffers and the size of each buffer for each transaction on the send queue:

```
SELECT cbkeyserverid,cbkeyid,cbkeypos,count(*),sum(cbsize)
    from syscdrsend_buf
    group by cbkeyserverid, cbkeyid, cbkeypos
    order by cbkeyserverid, cbkeyid, cbkeypos
```

All queues are present on all the replication servers, regardless of whether the replication server is a source or a target for a particular transaction. Some of the queues are always empty. For instance, the send queue on a target-only server is always empty.

Each queue is two-dimensional. Every queue has a list of transaction headers. Each transaction header in turn has a list of buffers that belong to that transaction.

Columns of the Transaction Tables

All the tables whose names end with **_txn** have the same columns and the same column definitions. The information in the tables represents *only* transactions in memory and not those spooled to disk.

The **ctstamp1** and **ctstamp2** columns combine to form the primary key for these tables.

Column	Type	Description
ctkeyserverid	integer	Server ID of the database server where this data originated This server ID is the group ID from the sqlhosts file or the SQLHOSTS registry key.
ctkeyid	integer	Logical log ID
ctkeypos	integer	Position in the logical log on the source server for the transaction represented by the buffer
ctkeysequence	integer	Sequence number for the buffer within the transaction
ctstamp1	integer	Together with **ctstamp2**, forms an insertion stamp that specifies the order of the transaction in the queue
ctstamp2	integer	Together with **ctstamp1**, forms an insertion stamp that specifies the order of the transaction in the queue
ctcommittime	integer	Time when the transaction represented by this buffer was committed
ctuserid	integer	Login ID of the user who committed this transaction
ctfromid	integer	Server ID of the server that sent this transaction Used only in hierarchical replication

Columns of the Buffer Tables

The tables whose names end with **_buf** give information about the buffers that form the transactions listed in the **_txn** table. All the **_buf** tables have the same columns and the same column definitions.

Column	Type	Description
cbflags	integer	Internal flags for this buffer
cbsize	integer	Size of this buffer in bytes
cbkeyserverid	integer	Server ID of the database server where this data originated This server ID is group ID from the sqlhosts file or the SQLHOSTS registry key.
cbkeyid	integer	Login ID of the user who originated the transaction represented by this buffer
cbkeypos	integer	Log position on the source server for the transaction represented by this buffer
cbkeysequence	integer	Sequence number for this buffer within the transaction
cbreplid	integer	Replicate identifier for the data in this buffer
cbcommittime	integer	Time when the transaction represented by this buffer was committed

The following columns combine to form the primary key for this table: **cbkeyserverid**, **cbkeyid**, **cbkeypos**, **cbkeysequence**.

The columns **cbkeyserverid**, **cbkeyid**, and **cbkeypos** form a foreign key that points to the transaction in the **_txn** table that the buffer represents.

Replication Examples

This appendix contains simple examples of replication using the *command-line utility* (CLU). Appendix A, "Command-Line Utility Reference" documents the CLU.

Replication Example Environment

To run the replication examples in this chapter, you need three Informix database servers. Each database server must be in a database server group.

This example uses the following environment:

- Three computers (**s1**, **s2**, and **s3**) are hosts for the database servers **usa**, **italy**, and **japan**, respectively. Each computer has active network connections to the other two computers.

- The database servers **usa**, **italy**, and **japan** are members of the database server groups **g_usa**, **g_italy**, and **g_japan**, respectively.

  For information about database server groups, see "Setting up Database Server Groups" on page 4-5.

UNIX

■ The UNIX **sqlhosts** file for each database server contains the following connectivity information.

```
g_usa          group      –          –              i=1
usa            ontlitcp   s1         techpubs1      g=g_usa
g_italy        group      –          –              i=8
italy          ontlitcp   s2         techpubs2      g=g_italy
g_japan        group      –          –              i=6
japan          ontlitcp   s3         techpub6       g=g_usa
```

♦

WIN NT/2000

See Appendix F, "SQLHOSTS Registry Key" for information on how to prepare the SQLHOSTS connectivity information using the information in the UNIX **sqlhosts** file. ♦

■ The ONCONFIG file contains the following CDR_QDATA_SBSPACE entry:

```
CDR_QDATA_SBSPACE sbspace #CDR queue smart blob space
```

You must create an sbspace for the row data and set the CDR_QDATA_SBSPACE parameter to the location of that sbspace. For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-11 and "CDR_QDATA_SBSPACE" on page B-9.

■ All commands in this example, except for creation of the sample databases on **italy** and **japan**, are issued from the computer **s1**.

■ The databases for the examples in this chapter are identical **stores_demo** databases with logging, as follows:

❑ Create a database named **stores** on the **usa** database server:

```
s1> dbaccessdemo -log stores
```

❑ Create a database named **stores** on the **italy** database server:

```
s1> rlogin s2
s2> dbaccessdemo -log stores
```

❑ Create a database named **stores** on the **japan** database server:

```
s1> rlogin s3
s2> dbaccessdemo -log stores
```

For information on preparing data for replication, see "Data Preparation Example" on page 4-24.

Primary-Target Example

This section contains a simple example of *primary-target* replication. In primary-target replication, only changes to the primary table are replicated to the other tables in the replicate. Changes to the secondary tables are not replicated.

Preparing for a Primary-Target Replication

In this example, define the **g_usa** and **g_italy** database server groups as Enterprise Replication servers and create a replicate, **repl1**.

To define the database server groups and create the replicate

1. Create and populate the database that defines the **usa** database server as a replication server:

   ```
   cdr define server --init g_usa
   ```

   Before replicating data, you must define the database servers as *replication servers.* A replication server is a database server that has an extra database that holds replication information.

   The --**init** option specifies that this server is a new replication server. When you define a replication server, you *must* use the name of the database server group (**g_usa**) rather than the database server name.

2. Display the replication server that you defined to verify that the definition succeeded:

   ```
   cdr list server
   ```

   The command returns the following information:

   ```
   SERVER      ID STATE     STATUS     QUEUE    CONNECTION CHANGED
   -----------------------------------------------------------
   g_usa        1 Active    Local      0
   ```

3. Define the second database server, **italy**, as a replication server:

```
cdr define server --connect=italy --init \
--sync=g_usa g_italy
```

The --**connect** option allows you to define **italy** (on the **s2** computer) while working at the **s1** (**usa**) computer. The --**sync** option instructs the command to use the already-defined replication server (**g_usa**) as a pattern for the new definition. The --**sync** option also links the two replication servers into a replication environment.

Tip: In all options except the --connect option, Enterprise Replication uses the name of the database server group to which a database server belongs, instead of the name of the database server itself.

4. Verify that the second definition succeeded:

```
cdr list server
```

The command returns the following information:

```
SERVER   ID   STATE        STATUS     QUEUE    CONNECTION CHANGED
-------------------------------------------------------------------
g_italy  8    Active       Connected  0        JUN 14 14:38:44 2000
g_usa    1    Active       Local      0
```

5. Define the replicate **repl1**:

```
cdr define replicate --conflict=ignore repl1 \
"P stores@g_usa:informix.manufact" \
"select * from manufact" \
"R stores@g_italy:informix.manufact" \
"select * from manufact"
```

These lines are all one command. The backslashes (\) at the end of the lines indicate that the command continues on the next line.

This step specifies that conflicts should be ignored and describes two participants, **usa** and **italy**, in the replicate:

■　The P indicates that in this replicate **usa** is a primary server. That is, if any data in the selected columns changes, that changed data should be sent to the secondary servers.

■　The R indicates that in this replicate **italy** is a secondary server (receive-only). The specified table and columns receive information that is sent from the primary server. Changes to those columns on **italy** are *not* replicated.

6. Display the replicate that you defined, so that you can verify that the definition succeeded:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   ----------------------------------------------
   REPLICATE:      repl1
   STATE:          Inactive
   CONFLICT:       Ignore
   FREQUENCY:      immediate
   QUEUE SIZE:     0
   PARTICIPANT:    g_usa:informix.manufact
                   g_italy:informix.manufact
   ```

   Step 5 *defines* a replicate but does not make the replicate active. The output of step 6 shows that the STATE of the replicate is INACTIVE.

7. Make the replicate active:

   ```
   cdr start replicate repl1
   ```

8. Display the replicate so that you can verify that the STATE has changed to ACTIVE:

   ```
   cdr list replicate
   ```

   The command returns the following information:

   ```
   CURRENTLY DEFINED REPLICATES
   ----------------------------------------------
   REPLICATE:      repl1
   STATE:          Active
   CONFLICT:       Ignore
   FREQUENCY:      immediate
   QUEUE SIZE:     0
   PARTICIPANT:    g_usa:informix.manufact
                   g_italy:informix.manufact
   ```

   If any changes are made to the **manufact** table, the changes will be replicated to the other participants in the replicate.

Cause a Replication

Now you can modify the **manufact** table on the **usa** database server and see the change reflected in the **manufact** table on the **italy** database server.

To cause a replication

1.   Use DB-Access to insert a value into the **manufact** table on **usa**:

```
INSERT INTO stores@usa:manufact \
VALUES ('AWN','Allwyn','8');
```

2.   Observe the changes on **usa** and on **italy**:

```
SELECT * from stores@usa:manufact
SELECT * from stores@italy:manufact
```

Do Not Cause a Replication

In **repl1**, **usa** is the primary database server and **italy** is the target. Changes made to the **manufact** table on **italy** do not replicate to **usa**.

To not cause a replication

1.   Use DB-Access to insert a value into the **manufact** table on **italy**:

```
INSERT INTO stores@italy:manufact \
VALUES ('ZZZ','Zip','9');
```

2.   Verify that the change occurred on **italy** but did not replicate to the **manufact** table on **usa**:

```
SELECT * from stores@usa:manufact
SELECT * from stores@italy:manufact
```

Update-Anywhere Example

This example builds on the previous example and creates a simple *update-anywhere* replication. In update-anywhere replication, changes to *any* table in the replicate are replicated to *all* other tables in the replicate. In this example, any change to the **stock** table of the **stores** database on any database server in the replicate will be replicated to the **stock** table on the other database servers.

Preparing for an Update-Anywhere Replication

In this example, define the **repl2** replicate.

To prepare for update-anywhere replication

1. Define the replicate, **repl2**:

    ```
    cdr define replicate --conflict=ignore repl2 \
    "stores@g_usa:informix.stock" "select * from stock" \
    "stores@g_italy:informix.stock" "select * from stock"
    ```

    These lines are all one command. The backslashes (\) at the end of
    the lines indicate that the command continues on the next line.

    This step specifies that conflicts should be ignored and describes two
    participants, **usa** and **italy** (including the table and the columns to
    replicate) in the replicate.

    Because neither P (primary) nor R (receive-only) is specified, the rep-
    licate is defined as update-anywhere. If any data in the selected
    columns changes, on either participant, that changed data should be
    sent to the other participants in the replicate.

2. Display all the replicates so that you can verify that your definition
    of **repl2** succeeded:

    ```
    cdr list replicate
    ```

    The command returns the following information:

    ```
    CURRENTLY DEFINED REPLICATES
    --------------------------------------------------------
    REPLICATE:      repl1
    STATE:          Active
    CONFLICT:       Ignore
    FREQUENCY:      immediate
    QUEUE SIZE:     0
    PARTICIPANT:    g_usa:informix.manufact
                    g_italy:informix.manufact

    REPLICATE:      repl2
    STATE:          Inactive
    CONFLICT:       Ignore
    FREQUENCY:      immediate
    QUEUE SIZE:     0
    PARTICIPANT:    g_usa:informix.stock
                    g_italy:informix.manufact
    ```

    Although this output shows that **repl2** exists, it does not show the
    participant modifiers (the SELECT statements) for **repl2**.

3. Display the participant modifiers for **repl2**:

```
cdr list replicate repl2
```

This command returns the following information:

```
REPLICATE    TABLE                           SELECT
------------------------------------------------------------
repl2        stores@g_usa:informix.stock     select * from stock
repl2        stores@g_italy:informix.stock   select * from stock
```

4. Add the **japan** database server to the replication system already defined in the previous example:

```
cdr define server --connect=japan --init \
--sync=g_usa g_japan
```

You can use either **g_usa** or **g_italy** in the --**sync** option.

Enterprise Replication maintains identical information on all servers that participate in the replication system. Therefore, when you add the **japan** database server, information about that server is propagated to all previously-defined replication servers (**usa** and **italy**).

5. Display the replication servers so that you can verify that the definition succeeded:

```
cdr list server
```

The command returns the following information:

```
SERVER  ID  STATE     STATUS      QUEUE   CONNECTION CHANGED
--------------------------------------------------------------------
g_italy 8   Active    Connected 0         JUN 14 14:38:44 2000
g_japan 6   Active    Connected 0         JUN 14 14:38:44 2000
g_usa   1   Active    Local
```

6. Add the participant and participant modifier to **repl2**:

```
cdr change replicate --add repl2 \
"stores@g_japan:informix.stock" "select * from stock"
```

7. Display detailed information about **repl2** after adding the participant in step 6:

```
cdr list replicate repl2
```

The command returns the following information:

```
REPLICATE     TABLE                           SELECT
-----------------------------------------------------------
repl2      stores@g_usa:informix.stock      select * from stock
repl2      stores@g_italy:informix.stock    select * from stock
repl2      stores@g_japan:informix.stock    select * from stock
```

8. Make the replicate active:

```
cdr start replicate repl2
```

Changes made to the **stock** table on **usa** are reflected in the **stock** tables on both **italy** and **japan**.

9. Display a list of replicates so that you can verify that the STATE of **repl2** has changed to ACTIVE

```
cdr list replicate
```

The command returns the following information:

```
CURRENTLY DEFINED REPLICATES
------------------------------------------------------
REPLICATE:     repl1
STATE:         Active
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   g_usa:informix.manufact
               g_italy:informix.manufact

REPLICATE:     repl2
STATE:         Active
CONFLICT:      Ignore
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   g_usa:informix.stock
               g_italy:informix.manufact
               g_japan:informix.manufact
```

Cause a Replication

Now you can modify the **stock** table on one database server and see the change reflected on the other database servers.

To cause a replication

1. Use DB-Access to insert a line into the **stock** table on **usa**:

```
INSERT INTO stores@usa:stock VALUES (401, "PRC", "ski boots",
200.00, "pair", "pair");
```

2. Observe the change on the **italy** and **japan** database servers:

```
SELECT * from stores@italy:stock;
SELECT * from stores@japan:stock;
```

To update the stock table on the japan database server

1. Use DB-Access to change a value in the **stock** table on **japan**:

```
UPDATE stores@japan:stock SET unit_price = 190.00
WHERE stock_num = 401;
```

2. Verify that the change has replicated to the **stock** table on **usa** and on **italy**:

```
SELECT * from stores@usa:stock WHERE stock_num = 401;
SELECT * from stores@italy:stock WHERE stock_num = 401;
```

Hierarchy Example

The example in this section adds a replication tree to the fully-connected environment of the **usa**, **italy**, and **japan** replication servers. The nonroot servers **boston** and **denver** are children of **usa**. (The leaf server **miami** is a child of **boston**.) Figure E-1 shows the replication hierarchy.
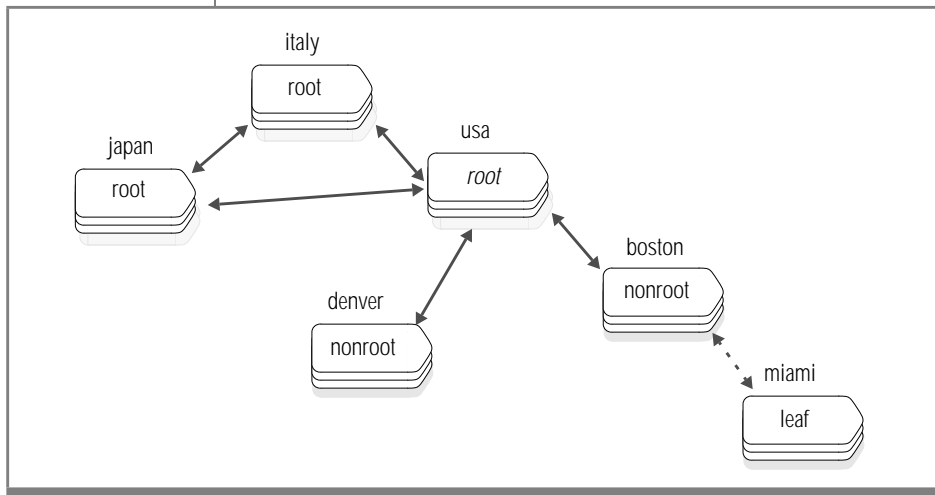
Preparing for a Hierarchy Example

To try this example, you need to prepare three additional database servers: **boston**, **denver**, and **miami**. To prepare the database servers, use the techniques described in "Replication Example Environment" on page E-1.

Defining a Hierarchy

The following example defines a replication hierarchy that includes **denver**, **boston**, and **miami** and whose root is **usa**.

To define a hierarchy

1. Add **boston** to the replication hierarchy as a nonroot server attached to the root server **usa**:

   ```
   cdr define server --connect boston --nonroot --init \
   --sync g_usa g_boston
   ```

   The backslash (\) indicates that the command continues on the next line.

2. Add **denver** to the replication hierarchy as a nonroot server attached to the root server **usa**:

   ```
   cdr define server -c denver -I -N --ats=/ix/myats \
   -S g_usa g_denver
   ```

   This command uses short forms for the **connect**, **init**, and **sync** options. (For information about the short forms, refer to "Option Abbreviations" on page A-84.) The command also specifies a directory for collecting information about failed replication transactions, **/ix/myats**.

3. List the replication servers as seen by the **usa** replication server:

   ```
   cdr list server
   ```

   The root server **usa** is fully connected to all the other root servers. Therefore **usa** knows the connection status of all other root servers and of its two child servers, **denver** and **boston**. The command returns the following information:

```
SERVER     ID   STATE       STATUS      QUEUE   CONNECTION CHANGED
--------------------------------------------------------------------
g_boston   3    Active      Connected   0       Aug 19 14:20:03 2000
g_denver   27   Active      Connected   0       Aug 19 14:20:03 2000
g_italy    8    Active      Connected   0       Aug 19 14:20:03 2000
g_japan    6    Active      Connected   0       Aug 19 14:20:03 2000
g_usa      1    Active      Local       0
```

4. List the replication servers as seen by the **denver** replication server:

```
cdr list server --connect denver
```

The nonroot server **denver** has a complete global catalog of replication information, so it knows all the other servers in its replication system. However, **denver** knows the connection status only of itself and its parent, **usa**.

The command returns the following information:

```
SERVER    ID   STATE        STATUS      QUEUE   CONNECTION CHANGED
----------------------------------------------------------------------
g_boston 3    Active                   0
g_denver 27   Active       Local       0
g_italy  8    Active                   0
g_japan  6    Active                   0
g_usa    1    Active       Connected   0       Aug 19 14:20:03 2000
```

5. Define **miami** as a leaf server whose parent is **boston**:

```
cdr define server -c miami -I --leaf -S g_boston g_miami
```

6. List the replication servers as seen by **miami**:

```
cdr list server -c miami
```

As a leaf replication server, **miami** has a limited catalog of replication information. It knows only about itself and its parent.

The command returns the following information:

```
SERVER    ID   STATE        STATUS      QUEUE   CONNECTION CHANGED
----------------------------------------------------------------------
g_boston 3    Active       Connected   0       Aug19 14:35:17 2000
g_miami  4    Active       Local       0
```

7. List details about the **usa** replication server:

```
cdr list server g_usa
```

The server is a *hub*; that is, it forwards replication information to and from other servers. It uses the default values for idle timeout, send queue, receive queue, and ATS directory.

The command returns the following information:

```
NAME      ID    ATTRIBUTES
----------------------------------------------------------------------
g_usa    1     timeout=15 hub sendq=rootdbs recvq=rootdbs atsdir=/tmp
```

WIN NT/2000

SQLHOSTS Registry Key

When you install the database server, the **setup** program creates the following key in the Windows registry:

```
HKEY_LOCAL_MACHINE\SOFTWARE\INFORMIX\SQLHOSTS
```

This branch of the HKEY_LOCAL_MACHINE subtree stores the **sqlhosts** information. Each key on the SQLHOSTS branch is the name of a database server. When you click the database server name, the registry displays the values of the HOST, OPTIONS, PROTOCOL, and SERVICE fields for that particular database server.

Each computer that hosts a database server or a client must include the connectivity information either in the SQLHOSTS registry key or in a central registry. When the client application runs on the same computer as the database server, they share a single SQLHOSTS registry key.

The Location of the SQLHOSTS Registry Key

When you install the database server on Windows, the installation program asks where you want to store the SQLHOSTS registry key. You can specify one of the following two options:

- The local computer where you are installing the database server

- Another computer in the network that serves as a central, shared repository of **sqlhosts** information for multiple database servers in the network

Local SQLHOSTS Registry Key

If you use the SQLHOSTS registry key on the local computer, for all database servers, the correct SQLHOSTS registry key must exist on *all* computers involved in replication. In addition, the **hosts** and **services** files on each computer must include information about all database servers.

For example, to set up replication between the server instance **srv1** on the computer **host1** and the server instance **srv2** on **host2**, you must ensure the following:

- Both **host1** and **host2** include SQLHOSTS registry key entries for **srv1** and **srv2**.
- The **services** file on both computers include the details of the services used by both database server instances.

Shared SQLHOSTS Registry Key

If you use a shared SQLHOSTS registry key you do not need to maintain the same **sqlhosts** information on multiple computers. However, the **hosts** and **services** files on *each* computer must contain information about all computers that have database servers.

If you specify a shared SQLHOSTS registry key, you must set the environment variable **INFORMIXSQLHOSTS** on your local computer to the name of the Windows computer that stores the registry. The database server first looks for the SQLHOSTS registry key on the INFORMIXSQLHOSTS computer. If the database server does not find an SQLHOSTS registry key on the INFORMIX-SQLHOSTS computer, or if **$INFORMIXSQLHOSTS** is not set, the database server looks for an SQLHOSTS registry key on the local computer.

You must comply with Windows network-access conventions and file permissions to ensure that the local computer has access to the shared SQLHOSTS registry key. For information about network-access conventions and file permissions, see your Windows documentation.

Preparing the SQLHOSTS Connectivity Information

Preparing the SQLHOSTS connectivity information consists of setting up registry keys on each computer that hosts a database server that participates in a replicate.

To prepare the SQLHOSTS connectivity information

1. Set up the SQLHOSTS registry key on the local computer.
2. Set up the database server group registry key on the local computer.

   See "Setting Up the Database Server Group Registry Key" on page F-6.
3. Set up the SQLHOSTS and group registry keys on all computers that are participants in the replicate.

   See "Setting up the Registry Keys on All Computers" on page F-8.
4. Ensure that the services files on each computer include entries for all database servers that are participants in the replicate.

   See "Verifying the services Files on All Computers" on page F-8.

Setting up the SQLHOSTS Registry with ISA

Important: *Informix strongly recommends that you use Informix Server Administrator (ISA), rather than* **regedt32***, to set up the SQLHOSTS registry key and database server group registry key on your Windows system. *In addition, ISA allows you to administer your replication system from a web browser. For more information, see the Documentation Notes described in "Additional Documentation" on page 15 of the Introduction.*

See the online help for ISA for details on setting up the SQLHOSTS registry key and database server group registry key.

Setting up the SQLHOSTS Registry Key with regedt32

Informix recommends that you use ISA to set up the SQLHOSTS registry key.

Important: *Use extreme caution with regedt32. If you make mistakes when editing the registry, you can destroy the configuration, not only of your Informix products, but of your other applications.*

To set up SQLHOSTS with regedt32

1. Run the Windows program, **regedt32**.

2. In the Registry Editor window, select the window for the HKEY_LOCAL_MACHINE subtree.

3. Click the folder icons to select the \SOFTWARE\INFORMIX\ SQLHOSTS branch.

4. With the SQLHOSTS key selected, choose **Edit→Add Key**.

5. In the Add Key dialog box, type the name of the database server in the Key Name dialog box.

   Leave the Class dialog box blank. Click **OK**.

6. Select the new key that you just made (the key with the database server name).

7. Choose **Edit→Add Value.**

8. In the Add Value dialog box, type one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, SERVICE) in the Value Name dialog box.

   Do not change the Data Type box. Click **OK**.

9. In the String Editor dialog box, type the value for the field that you selected and click **OK**.

10. Repeat steps 8 and 9 for each field of the **sqlhosts** information.

Figure F-1 illustrates the location and contents of the SQLHOSTS registry key.

Figure F-1
sqlhosts Information in the Windows Registry
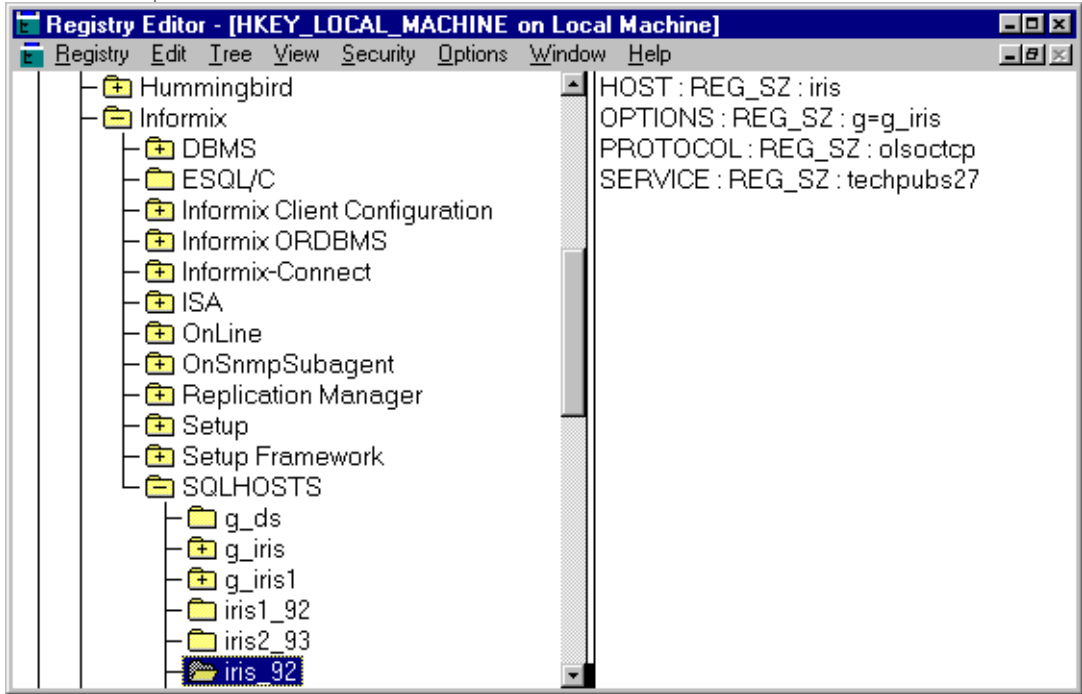
Setting Up the Database Server Group Registry Key

After you create the registry key for the database server, you must make a registry key for the database server group that includes the database server. For more information, refer to "Verifying SQLHOSTS" on page 4-5.

To set up the database server group registry key

1. With the SQLHOSTS key selected, choose **Edit→Add Key**.

2. In the Add Key dialog box, type the name of the database server group in the Key Name dialog box.

    This value must correspond to the OPTIONS value in the database server name key.

Tip: In this manual (and in Figure F-1 on page F-5), each of the names of the database server groups are the database server names prefixed by g_. The g_ prefix is not a requirement; it is just the convention that this manual uses.

    Leave the Class dialog box blank. Click **OK**.

3. Select the new key that you just made (the key with the database server group name).

4. Choose **Edit→Add Value.**

5. In the Add Value dialog box, type one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, SERVICE) in the Value Name dialog box.

    Do not change the Data Type dialog box. Click **OK**.

6. In the String Editor dialog box, type the value for the field that you selected and click **OK**.

    For a database server group, enter the following values:

    ```
    HOST        -
    OPTIONS     i=unique-integer-value
    PROTOCOL    group
    SERVICE     -
    ```

    Each database server group must have an associated identifier value (**i**=) that is unique among all database servers in your environment. Enter a minus (-) for HOST and SERVICE to indicate that you are not assigning specific values to those fields.

7. Repeat steps 6 and 7 for each field of the **sqlhosts** information.

    Figure F-2 illustrates the contents of the database server group regis-
    try key after you add the values to the keys.
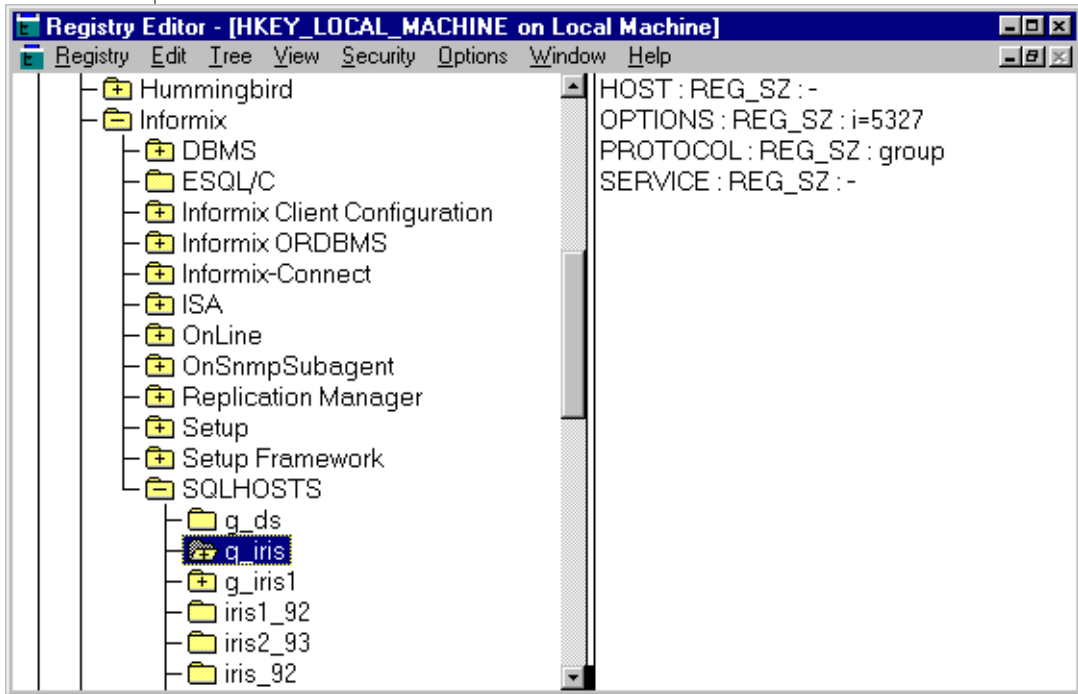
8. Exit from the Registry Editor.

Setting up the Registry Keys on All Computers

Now, update the registry keys on all computers that participate in replication.

To update the registry keys on all computers

1. Set up the SQLHOSTS registry key on all computers that participate in replication.

    See "Setting up the SQLHOSTS Registry Key with regedt32" on page F-4.

2. Set up the database server group registry key on all computers that participate in replication.

    See "Setting Up the Database Server Group Registry Key" on page F-6.

Verifying the services Files on All Computers

Finally, on each computer that participates in replication, make sure that the **services** file (located in the **C:\WINNT/system32/drivers/etc/** directory) contains entries for all the database servers.

To verify the services files on all computers

1. Check the **services** file on the first host (for example, **host1**).

    The file might look like this

    ```
    techpubs27     4599/tcp      # service for online instance denver
    techpubs28     4600/tcp      # service for online instance boston
    ```

2. Check the **services** file on the second host (for example, **host2**).

    The file might should look the same as the file on **host1**:

    ```
    techpubs27     4599/tcp      # service for online instance denver
    techpubs28     4600/tcp      # service for online instance boston
    ```

Index