Informix Product Family
Informix
Version 11.70

IBM Informix Enterprise Replication Guide

SC27-3539-04

Informix Product Family
Informix
Version 11.70

IBM Informix Enterprise Replication Guide

SC27-3539-04

Note
Before using this information and the product it supports, read the information in “Notices” on page L-1.

This edition replaces SC27-3539-03.

Contents

Part 3. Appendixes

Introduction

About this publication

This publication describes IBM® Informix® Enterprise Replication and the concepts of data replication. This publication 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 publication 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

Assumptions About Your Locale

IBM 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 publication 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 *IBM Informix GLS User's Guide*.

Demonstration Databases

The DB-Access utility, which is provided with your IBM 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 IBM Informix publications 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 *IBM Informix DB–Access User's Guide*. For descriptions of the databases and their contents, see the *IBM 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.

What's New in Enterprise Replication for Informix, Version 11.70

This publication includes information about new features and changes in existing functionality.

For a complete list of what's new in this release, see the release notes or the information center at http://publib.boulder.ibm.com/infocenter/idshelp/v117/topic/com.ibm.po.doc/new_features.htm.

Table 1. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC5

Overview	Reference
Replication errors on leaf nodes Because leaf servers have limited information about other replication servers, the **syscdrerror** tables on leaf nodes no longer contain replication errors for other replication servers.	"cdr error" on page A-91

Table 2. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC4

Overview	Reference
Configure secure connections for replication servers You can use the connection security option in the `sqlhosts` file and create an encrypted password file so that the Connection Manager and the CDR utility can securely connect to networked servers. Use the **onpassword** utility to encrypt or decrypt a password file.	"Configuring secure connections for replication servers" on page 4-4
Easier setup of faster consistency checking When you increase the speed of consistency checking by creating an index on the **ifx_replcheck** shadow column, you no longer need to include the conflict resolution shadow columns in the replicated table. In the CREATE TABLE statement, the WITH REPLCHECK keywords do not require the WITH CRCOLS keywords.	"Indexing the ifx_replcheck Column" on page 8-19

Table 3. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC3

Overview	Reference
Automatically connecting to a grid You can connect to a grid automatically by including the **ifx_grid_connect()** procedure as part of the **sysdbopen()** procedure. Set the value of the *ER_enable* option of the **ifx_grid_connect()** procedure to 2 or 3 to suppress errors that would prevent the session from accessing the database.	"ifx_grid_connect() procedure" on page C-1

Table 3. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC3 (continued)

Overview	Reference
Code set conversion for Enterprise Replication Enterprise Replication supports replication between database servers that use different code sets. You can convert servers to the Unicode code set with minimal application downtime, convert servers from one code set to another, and replicate data between servers in different locales. You can enable replication between code sets by using the UTF8 option when creating the replicate definition.	"Enabling code set conversion between replicates" on page 6-14

Table 4. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC1

Overview	Reference
New editions and product names IBM Informix Dynamic Server editions were withdrawn and new Informix editions are available. Some products were also renamed. The publications in the Informix library pertain to the following products: • IBM Informix database server, formerly known as IBM Informix Dynamic Server (IDS) • IBM OpenAdmin Tool (OAT) for Informix, formerly known as OpenAdmin Tool for Informix Dynamic Server (IDS) • IBM Informix SQL Warehousing Tool, formerly known as Informix Warehouse Feature	For more information about the Informix product family, go to http://www.ibm.com/software/data/informix/.
Create a new replication domain by cloning a server You can create a new replication domain by cloning a server and then converting the two Informix database servers to replication servers. Use cloning and conversion if you want to set up replication based on the data on a source server that is not yet running Enterprise Replication. Because the source server does not have Enterprise Replication defined, you use the **ifxclone** utility to create a cluster containing a primary server and remote stand-alone (RS) secondary server. Use the **cdr start sec2er** command to convert the cluster to a pair of replication servers in a new domain.	"Creating a new domain by cloning a server" on page 6-2 "cdr start sec2er" on page A-137
Add a server to a replication domain by cloning You can add a replication server to an existing replication domain by using the ifxclone utility to clone an existing replication server onto a target database server.	"Adding a server to the domain by cloning a server" on page 6-5
Automating application connections to Enterprise Replication servers You can use the Connection Manager to direct application requests to the appropriate Enterprise Replication server. If you have created tables through a grid with replication enabled, you can route client connections to Enterprise Replication servers based on the quality of replicated data and transaction latency.	"Routing client connections in a grid" on page 7-16 "Routing client connections for a replicate set" on page 8-9

Table 4. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC1 (continued)

Overview	Reference
Replicate tables without primary keys If you do not want to have a primary key, or want to be able to update the primary key, on tables replicated by Enterprise Replication, you can use the ERKEY shadow columns in place of a primary key. If you create a replicated table through a grid, the ERKEY shadow columns are created automatically.	"Preparing tables without primary keys" on page 4-21
Set up and manage a grid for a replication domain You can create a grid of interconnected replication servers in a domain. You can use the grid to easily administer database servers in a replication domain. When you run the following types of commands from the grid, the commands are replicated to all servers in the grid: • Database schema updates • Administrative tasks • Routines	Chapter 7, "Grid setup and management," on page 7-1 Appendix C, "Grid routines," on page C-1
Control Enterprise Replication capture from within a transaction You can control which statements within a replicated transaction are replicated by using the **ifx_set_erstate()** procedure.	"Enabling replication within a grid transaction" on page 7-14 "Recapture replicated transactions" on page 8-28
Handle a potential log wrap situation in Enterprise Replication You can configure what actions occur automatically if a potential log wrap situation is detected during replication. A potential log wrap situation occurs when the log processing by Enterprise Replication lags behind the entries in the current log so that the Enterprise Replication replay position might be overwritten. In previous releases you could add logical logs or Enterprise Replication would block user sessions until the potential for log wrap diminished. Specify one or more of the following actions, in prioritized order, with the CDR_LOG_LAG_ACTION configuration parameter: • Compress logical logs in a staging directory • Add logical logs dynamically • Prevent blocking user sessions, but potentially overwrite the replay position • Block user sessions (default) • Shut down Enterprise Replication	"Handle potential log wrapping" on page 9-15 "CDR_LOG_LAG_ACTION Configuration Parameter" on page B-5
Improved Enterprise Replication error code descriptions Enterprise Replication return code documentation now includes descriptions and user actions.	"Return Codes for the cdr Utility" on page A-8

Table 4. What's New in IBM Informix Enterprise Replication Guide for 11.70.xC1 (continued)

Overview	Reference
Repair replication inconsistencies by time stamp If you have a replication domain with multiple master servers and your conflict resolution rule is time stamp or delete wins, you can repair inconsistencies based on the latest time stamps. In previous releases, you chose a master server to act as the correct version of the data and the repair made all the other participants' data match the master server's data. To repair by time stamp, use the **cdr check replicate** or **cdr check replicateset** commands with the **--repair** and **--timestamp** options and omit the **--master** option.	"Repair inconsistencies by time stamp" on page 8-20 "cdr check replicate" on page A-37 "cdr check replicateset" on page A-47
Temporarily disable an Enterprise Replication server You can temporarily stop replicating data to and from a replication server by using the **cdr disable server** command. The replication server stops queuing and receiving replicated data. Other replication servers in the replication domain also stop queuing data to the disabled replication server. However, because deleted row information on the disabled replication server is saved, you can quickly and accurately synchronize the data with a time stamp repair when you run the **cdr enable server** command.	"Temporarily stopping replication on a server" on page 8-3 "cdr disable server" on page A-86
Synchronize all replicates simultaneously If you want to repair all replicates in an Enterprise Replication domain, whether or not they are in a replicate set, use the **cdr check replicatset --repair** or the **cdr sync replicateset** command with the **--allrepl** option. The **--allrepl** and **--replset** options cannot be used together.	"cdr check replicateset" on page A-47 "cdr sync replicateset" on page A-164
Easier event alarm handling Event alarms now have a unique identification number for each specific message. You can write scripts to handle event alarms based on the unique identification number that corresponds to each specific message in an alarm class. Previously, event alarm handling scripts had to combine the class ID and the specific message.	"Enterprise Replication Event Alarms" on page 9-21
Repair jobs are deprecated The commands associated with repair jobs, such as **cdr define repair** and **cdr start repair**, are no longer valid. To repair inconsistencies, use the **cdr check replicate** or **cdr check replicateset** command with the **--repair** option.	

Example code conventions

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM 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 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. If you are using DB–Access, you must delimit multiple statements with semicolons.

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 documentation for your product.

Additional documentation

Documentation about this release of IBM Informix products is available in various formats.

You can access or install the product documentation from the Quick Start CD that is shipped with Informix products. To get the most current information, see the Informix information centers at ibm.com®. You can access the information centers and other Informix technical information such as technotes, white papers, and IBM Redbooks® publications online at http://www.ibm.com/software/data/sw-library/.

Compliance with industry standards

IBM Informix products are compliant with various standards.

IBM 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 IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

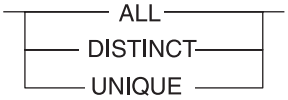
The IBM Informix Geodetic DataBlade® Module supports a subset of the data types from the *Spatial Data Transfer Standard (SDTS)—Federal Information Processing Standard 173*, as referenced by the document *Content Standard for Geospatial Metadata*, Federal Geographic Data Committee, June 8, 1994 (FGDC Metadata Standard).

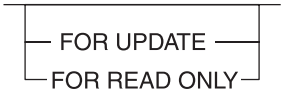
Syntax diagrams

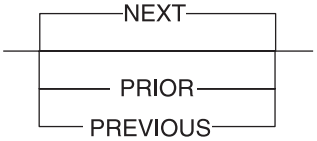
Syntax diagrams use special components to describe the syntax for statements and commands.

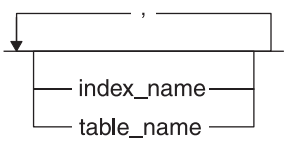
Table 5. Syntax Diagram Components

Component represented in PDF	Component represented in HTML	Meaning
	`>>----------`	Statement begins.
	`---------->`	Statement continues on next line.
	`>----------`	Statement continues from previous line.
	`----------><`	Statement ends.
	`--------SELECT----------`	Required item.
	`--+-----------------+---` `  '------LOCAL------'`	Optional item.

	`---+-----ALL-------+---`
	`   +--DISTINCT-----+`
	`   '---UNIQUE------'`
		Required item with choice. Only one item must be present.

	`---+------------------+---`
	`    +--FOR UPDATE-----+`
	`    '--FOR READ ONLY--'`
		Optional items with choice are shown below the main line, one of which you might specify.

	`    .---NEXT---------.`
	`----+----------------+---`
	`    +---PRIOR--------+`
	`    '---PREVIOUS-----'`
		The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line is used by default.

	`  .-------,-----------.`
	`  V                   |`
	`---+-----------------+---`
	`    +---index_name---+`
	`    '---table_name---'`
		Optional items. Several items are allowed; a comma must precede each repetition.

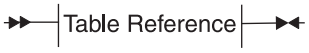
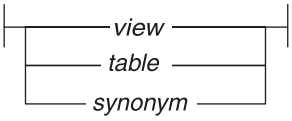
	`>>-| Table Reference |-><`	Reference to a syntax segment.

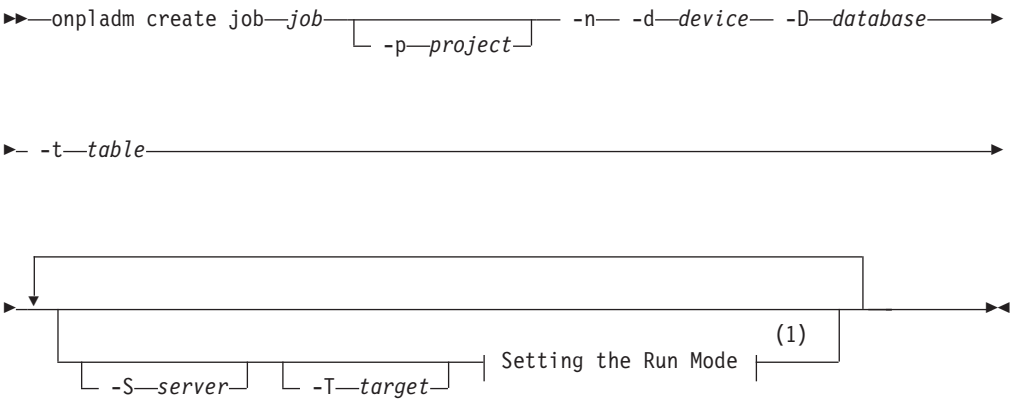
Table Reference
 
	`Table Reference`
	`|--+-----view--------+--|`
	`   +------table------+`
	`   '----synonym------'`
		Syntax segment.

How to read a command-line syntax diagram

Command-line syntax diagrams use similar elements to those of other syntax diagrams.

Some of the elements are listed in the table in Syntax Diagrams.

Creating a no-conversion job

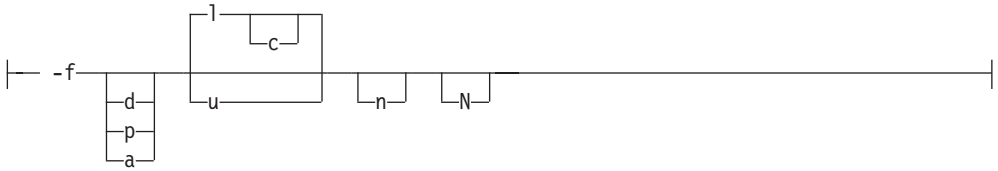


Notes:

1 See page Z-1

This diagram has a segment named "Setting the Run Mode," which according to the diagram footnote is on page Z-1. If this was an actual cross-reference, you would find this segment on the first page of Appendix Z. Instead, this segment is shown in the following segment diagram. Notice that the diagram uses segment start and end components.

Setting the run mode:



To see how to construct a command correctly, start at the upper left of the main diagram. Follow the diagram to the right, including the elements that you want. The elements in this diagram are case-sensitive because they illustrate utility syntax. Other types of syntax, such as SQL, are not case-sensitive.

The Creating a No-Conversion Job diagram illustrates the following steps:

1. Type **onpladm create job** and then the name of the job.
2. Optionally, type **-p** and then the name of the project.
3. Type the following required elements:
   - **-n**
   - **-d** and the name of the device
   - **-D** and the name of the database
   - **-t** and the name of the table

4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
   - **-S** and the server name
   - **-T** and the target server name
   - The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type **-f**, optionally type **d**, **p**, or **a**, and then optionally type **l** or **u**.
5. Follow the diagram to the terminator.

Keywords and punctuation

Keywords are words reserved for statements and all commands except system-level commands.

When a keyword appears in a syntax diagram, it is shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

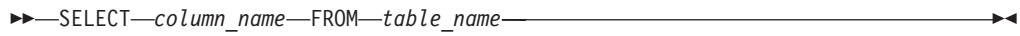
You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

Identifiers and names

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples.

You can replace a variable with an arbitrary name, identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase italic*.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.



When you write a SELECT statement of this form, you replace the variables *column_name* and *table_name* with the name of a specific column and table.

How to provide documentation feedback

You are encouraged to send your comments about IBM Informix user documentation.

Use one of the following methods:

- Send email to docinf@us.ibm.com.
- In the Informix information center, which is available online at http://www.ibm.com/software/data/sw-library/, open the topic that you want to comment on. Click the feedback link at the bottom of the page, fill out the form, and submit your feedback.

- Add comments to topics directly in the information center and read comments that were added by other users. Share information about the product documentation, participate in discussions with other users, rate topics, and more!

Feedback from all methods is monitored by the team that maintains the user documentation. The feedback methods are reserved for reporting errors and omissions in the documentation. For immediate help with a technical problem, contact IBM Technical Support at http://www.ibm.com/planetwide/.

We appreciate your suggestions.

Part 1. Introducing Enterprise Replication

These topics provide an overview of IBM Informix Enterprise Replication and how to administer it.

Chapter 1. About IBM Informix Enterprise Replication

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 IBM Informix Enterprise Replication and explains how this product replicates data.

IBM Informix Enterprise Replication

IBM Informix Enterprise Replication is an asynchronous, log-based tool for replicating data between IBM Informix 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.

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 to this, *synchronous* data replication replicates data 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 *IBM Informix Administrator's Guide*.

Asynchronous replication is often preferred because it allows for system and network failures.

Asynchronous replication allows the following replication models:

- Primary-target ("Primary-Target Replication System" on page 3-1)

  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 3-5)

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 3-6.

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 other 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 data source 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."

All Enterprise Replication operations are performed in parallel, which further extends the performance of Enterprise Replication.

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.

If high availability is critical, you can use high-availability clusters in conjunction with Enterprise Replication. High-availability clusters support synchronous data replication between database servers: a primary server, which can participate in Enterprise Replication, and one or more secondary servers, which do not participate in Enterprise Replication. If a primary server in a high-availability cluster fails, a secondary server can take over the role of the primary server, allowing it to participate in Enterprise Replication. Client connections to the original primary server can be automatically switched to the new standard server.

For more information on using high-availability clusters with Enterprise Replication, see Chapter 5, "Using High-Availability Clusters with Enterprise Replication," on page 5-1.

Consistent Information Delivery

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

IBM Informix 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 in a slightly different order on the target database.

If update conflicts occur, IBM Informix 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 3-6.

Repair and Initial Data Synchronization

Enterprise Replication provides initial data synchronization and multiple methods to repair replicated data.

You can easily bring a new table up-to-date with replication when you start a new replicate, or when you add a new participant to an existing replicate, by specifying an initial synchronization. Initial synchronization can be run online while replication is active.

If replication has failed for some reason, you can repair replicated data by running the **cdr sync replicate** or **cdr sync replicateset** command to resynchronize data and correct data mismatches between replicated tables. You can repair data while replication is active.

You can also repair data after replication has failed by using ATS and RIS files.Enterprise Replication examines the specified ATS or RIS file and attempts to reconcile the rows that failed to be applied.

Related concepts:

"Resynchronizing Data among Replication Servers" on page 8-14

Related tasks:

"Initially Synchronizing Data Among Database Servers" on page 6-20

"Repairing Failed Transactions with ATS and RIS Files" on page 8-21

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.

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.

You can add High-Availability Data Replication to any of these topologies.

Enterprise Replication supports all built-in IBM Informix data types, as well as extended and user-defined data types.

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

Enterprise Replication supports the Global Language Support (GLS) feature, which allows IBM Informix products to handle different languages, regional conventions, and code sets.

Related concepts:

"Primary-Target Replication System" on page 3-1

"Update-Anywhere Replication System" on page 3-5

"Choosing a Replication Network Topology" on page 3-15

"Enterprise Replication data types" on page 2-13

"Using GLS with Enterprise Replication" on page 2-13

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, "The cdr Command-Line Utility Reference," on page A-1.

Ease of Implementation

Enterprise Replication provides templates to allow easy set up and deployment of replication for clients with large numbers of tables to replicate. Administrators of Enterprise Replication can use templates to develop scripts and with only a few commands can set up replication over a large number of server nodes. Without using templates, many individual commands must be run. Using templates, you can also easily add a new server into your replication environment and optionally create and populate new database tables.

First, you create a template using the **cdr define template** command. This defines the database, tables, and columns and the characteristics of the replicates that will be created. You can view information about a template by using the **cdr list template** command from a non-leaf node.

Second, you instantiate the template on the servers where you want to replicate this data by running the **cdr realize template** command. If the table already exists on a node, Enterprise Replication verifies it matches the template definition. If the table does not exist on a node, Enterprise Replication can optionally create the table. Enterprise Replication can also optionally perform an initial data synchronization on all servers where you realize the template.

You can delete templates that you no longer need using the **cdr delete template** command.

See “Using Templates to Set Up Replication” on page 6-21 for more information. All replication commands mentioned in this section are described in detail in Appendix A, “The cdr Command-Line Utility Reference,” on page A-1.

Network Encryption

Enterprise Replication supports the same network encryption options that you can use with communications between server and clients to provide complete data encryption.

You can use the Secure Sockets Layer (SSL) protocol, a communication protocol that ensures privacy and integrity of data transmitted over the network, for connections between Enterprise Replication servers. For information on using the SSL protocol, see the "Secure Sockets Layer Communication Protocol" section of the *IBM Informix Security Guide*.

You can use encryption configuration parameters to provide data encryption with a standard cryptography library. A message authentication code (MAC) is transmitted as part of the encrypted data transmission to ensure data integrity. This is the same type of encryption provided by the ENCCSM communications support module for non-replication communication. Enterprise Replication shares the same ENCRYPT_CIPHERS, ENCRYPT_MAC, ENCRYPT_MACFILE, and ENCRYPT_SWITCH configuration parameters with high availability clusters. Enterprise Replication encryption configuration parameters are documented in Appendix B, “Enterprise Replication configuration parameter and environment variable reference,” on page B-1.

Enterprise Replication cannot accept a connection that is configured with a communications support module. To combine client/server network encryption with Enterprise Replication encryption, configure two network connections for each database server, one with CSM and one without. For more information, see “Configuring network encryption for replication servers” on page 4-5.

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 6-1. To define replicates, see "Defining Replicates" on page 6-7. Appendix H, "Replication Examples," on page H-1, has simple examples of declaring replication servers and defining replicates.

After you define the servers and replicates, Enterprise Replication replicates data in three phases:

1. "Data Capture" on page 1-7
2. "Data Transport" on page 1-12
3. "Applying Replicated Data" on page 1-12

The following diagram shows these three phases of replication and the Enterprise Replication components that perform each task.

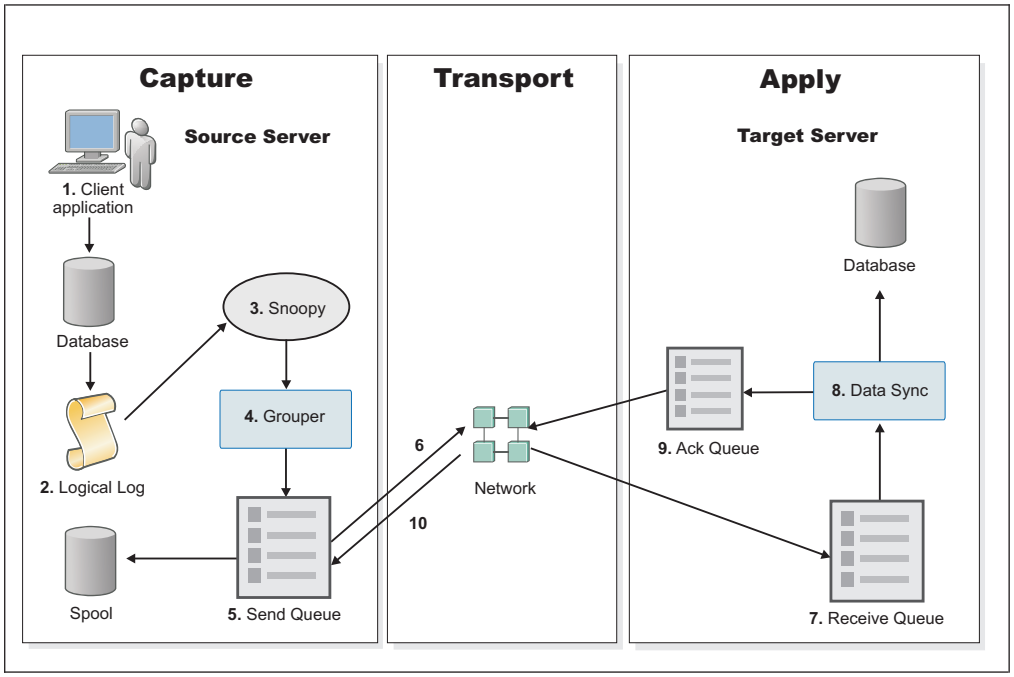


Figure 1-1. The Life Cycle of a Replicated Transaction

As shown in the diagram, the following process describes how Enterprise Replication replicates a transaction:

1. A client application performs a transaction in a database that is defined as a replicate.
2. The transaction is put into the logical log.
3. The log capture component, also known as the snoopy component, reads the logical log and passes the log records onto the grouper component.
4. The grouper component evaluates the log records for replication and groups them into a message that describe the operations that were in the original transaction.
5. The grouper component places the message in the send queue. Under certain situations, the send queue spools messages to disk for temporary storage.

6. The send queue transports the replication message across the Enterprise Replication network to the target server.
7. The replication message is placed in the receive queue at the target server.
8. The data sync component applies the transaction in the target database. If necessary, the data sync component performs conflict resolution.
9. An acknowledgment that the message was successfully applied is placed in the acknowledgment queue.
10. The acknowledgment message is sent back to the source server.

Data Capture

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.

IBM 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, by default, user transactions are blocked until Enterprise Replication advances. You can specify other responses to the potential for overwriting the Enterprise Replication replay position.

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

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.

Table 1-1 shows the logic that determines which rows are candidates for replication.

Table 1-1. Enterprise Replication Evaluation Logic

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

Table 1-1. Enterprise Replication Evaluation Logic (continued)

Initial Image	Replicate Evaluates	Final Image	Replicate Evaluates	Primary-Key Changed?	Send to Destination Database Server	Comments
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.

Table 1-1 on page 1-7 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 Table 1-1 on page 1-7 assumes that the source and the destination tables are initially synchronized (identical before replication begins). If the tables were 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 2-8.

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 IBM 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 "Replicating Simple and Smart Large Objects" on page 2-14, and "Replicating Opaque User-Defined Data Types" on page 2-16.

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, Enterprise Replication spools the receive queue transaction header and replicate records to a dbspace and the receive queue row data to an sbspace. For more information, see "Large Transactions" on page 2-11.

For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-10 and "Preventing Memory Queues from Overflowing" on page 9-14.

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 "Applying Replicated Data" on page 1-12).

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 8-1). 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 Data among Replication Servers" on page 8-14).

Data Evaluation Examples

Figure 1-2, Figure 1-3 on page 1-11, and Figure 1-4 on page 1-11 show three examples of how Enterprise Replication uses logic to evaluate transactions for potential replication.

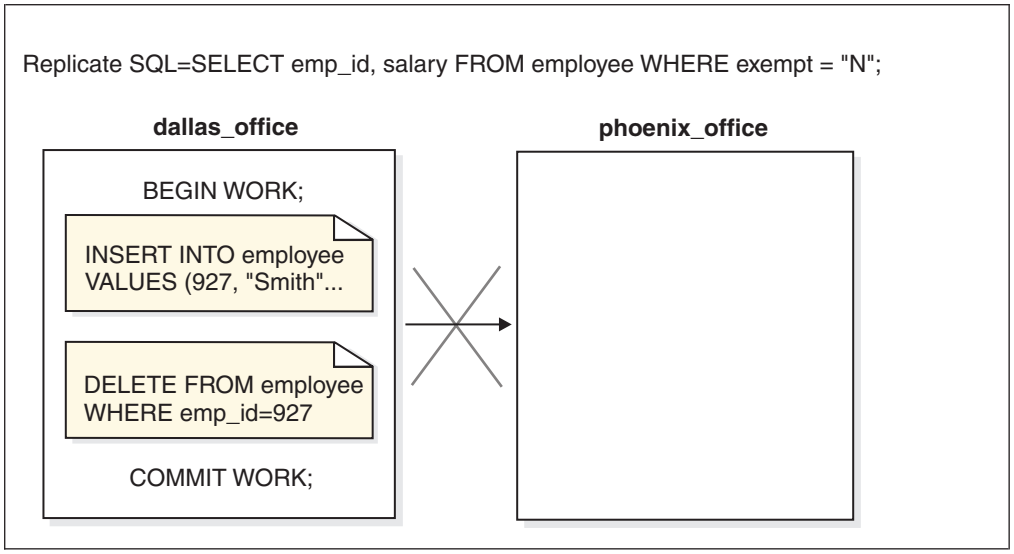


Figure 1-2. Insert Followed by a Delete

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

Table 1-2. 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 1-3 on page 1-11, Enterprise Replication uses the logic in Table 1-3 on page 1-11 to evaluate whether any information is sent to the destination database server.

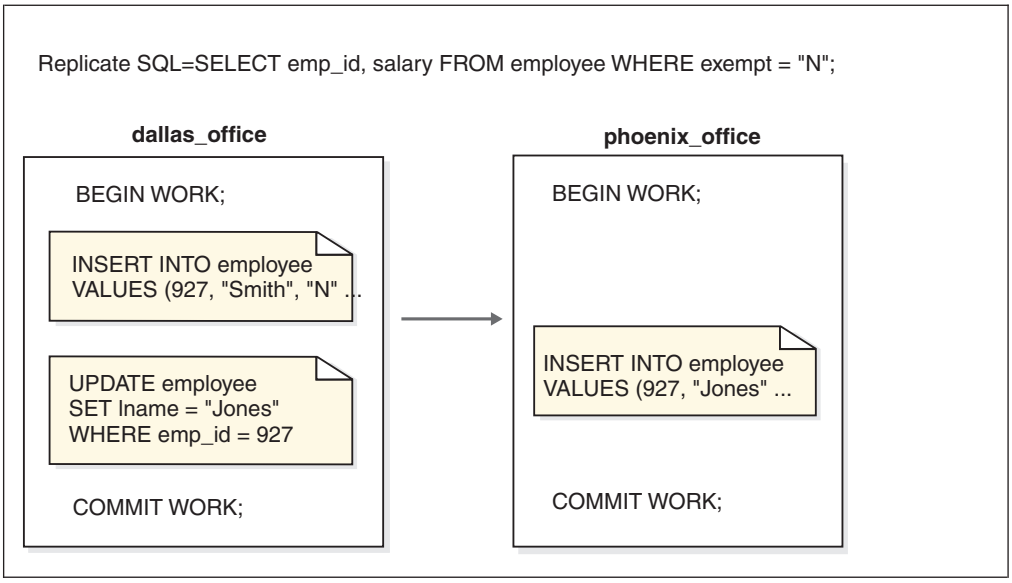


Figure 1-3. Insert Followed by an Update

Table 1-3. Insert Followed by An Update Evaluation Logic

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 are replicated where the exempt column contains a value of "N." 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 1-4, Enterprise Replication uses the logic in Table 1-4 on page 1-12 to evaluate whether any information is sent to the destination database server.

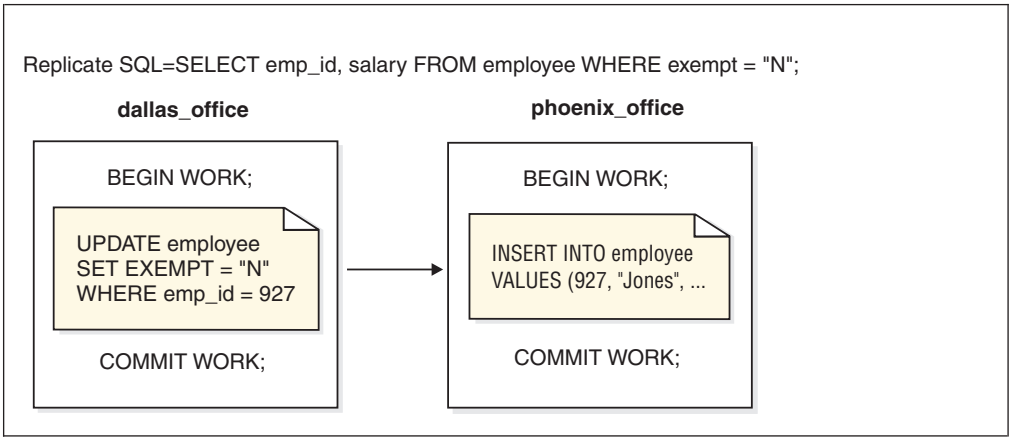


Figure 1-4. Update; Not Selected to Selected

Table 1-4. Update; Not Selected to Selected Evaluation Logic

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.

Data Transport

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.

Applying Replicated Data

IBM Informix 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. Data modifications resulting from synchronization, including modifications resulting from trigger invocation, are not replicated. The data-synchronization process ensures that transactions are applied at the target database servers in an order equivalent to the order that they were committed on the source database server. If consistency can be preserved, 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.

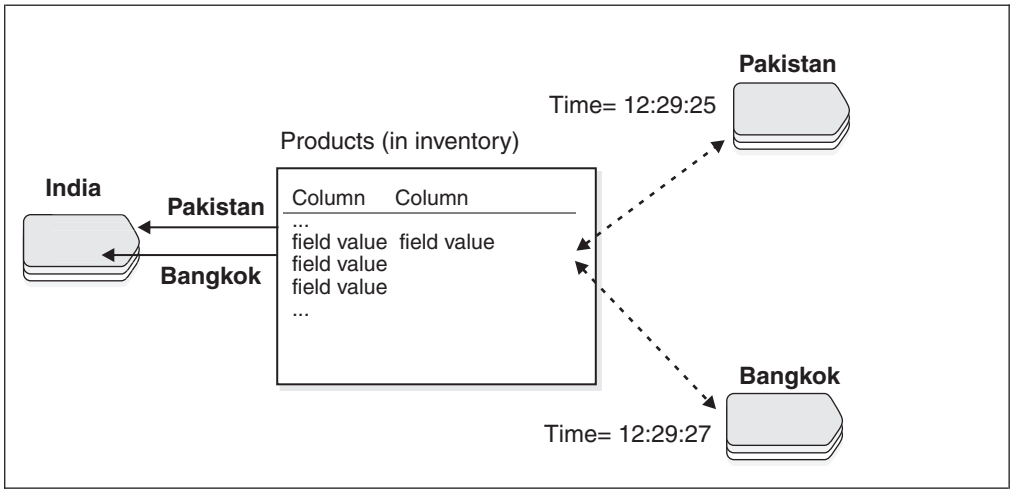


Figure 1-5. Collision Example

Figure 1-5 on page 1-12 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 3-6.

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

Chapter 2. Overview of Enterprise Replication Administration

In This Chapter

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

Setting Up Enterprise Replication

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," on page 3-1.
2. Prepare the replication environment.

   For information, see Chapter 4, "Preparing the Replication Environment," on page 4-1.
3. Initialize the database server.

   For information, see Chapter 6, "Defining Replication Servers, Replicates, Participants, and Replicate Sets," on page 6-1.
4. Define database servers for replication.

   For information, see Chapter 6, "Defining Replication Servers, Replicates, Participants, and Replicate Sets," on page 6-1.
5. Define replicates and participants.

   For information, see Chapter 8, "Managing Replication Servers and Replicates," on page 8-1.
6. Create replicate sets (optional).

   For information, see "Defining Replicate Sets" on page 6-18.
7. Start the replicate.

   For information, see Chapter 8, "Managing Replication Servers and Replicates," on page 8-1.

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

- "Managing Replication Servers" on page 8-1
- "Managing Replicates" on page 8-6
- "Managing Replicate Sets" on page 8-9
- Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1

Enterprise Replication Server Administrator

You need special privileges to run most Enterprise Replication commands.

The Enterprise Replication server administrator must have IBM Informix Database Server Administrator (DBSA) privileges to configure and manage Enterprise Replication or be user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Some command must be run as user **informix** or a member of the **Informix-Admin** group.

Related concepts:

"Interpreting the cdr Command-Line Utility Syntax" on page A-1

Related tasks:

"Defining Replication Servers" on page 6-1

Enterprise Replication Terminology

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

- Enterprise Replication server
- Replicate
- Master Replicate
- Shadow Replicate
- Participant
- Replicate Set
- Template
- Global Catalog

Enterprise Replication Server

An Enterprise Replication server, or *replication server*, is an IBM 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.

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 6-7 and "cdr define replicate" on page A-61.

Master Replicate

A *master replicate* is a replicate that guarantees data integrity by verifying that replicated tables on different servers have consistent column attributes. Master replicates also allow you to perform alter operations on replicated tables.

For more information, "Defining Master Replicates" on page 6-8 and "cdr define replicate" on page A-61.

Shadow Replicate

A *shadow replicate* is a copy of an existing (primary) replicate. Shadow replicates allow Enterprise Replication to manage alter and repair operations on replicated tables.

For more information, see "Defining Shadow Replicates" on page 6-9 and "cdr define replicate" on page A-61.

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 "Participant definitions" on page 6-7 and "Participant and participant modifier" on page A-4.

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 "Managing Replicate Sets" on page 8-9 and "cdr change replicateset" on page A-35.

Template

A *template* provides a mechanism to set up and deploy replication for a group of tables on one or more servers. This is especially useful if you have a large number of tables to replicate between many servers. Internally, a template defines a group of master replicates and a replicate set for a specified group of tables using attributes such as database, tables, columns and primary keys from the master node.

You create a template using the **cdr define template** command and then instantiate, or realize, it on servers with the **cdr realize template** command. See "Using Templates to Set Up Replication" on page 6-21 for more information.

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 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. For more information, see Table 3-5 on page 3-16.

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 8-1.

Leaf replication servers (Table 3-5 on page 3-16) have limited catalogs. 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 6-6.

Related concepts:

"Connect Option" on page A-3

Related tasks:

"Customizing the Replication Server Definition" on page 6-6

Enterprise Replication Considerations

These topics describe how Enterprise Replication interacts with other Informix functionality.

Asynchronous propagation considerations

Enterprise Replication asynchronously propagates many control operations through the Enterprise Replication network.

When you perform administrative functions using Enterprise Replication, the status that returns from those operations indicates 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. It might take a significant amount of time before the operation is propagated to database servers that are not connected to the Enterprise Replication network at all times.

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 was successfully propagated through the entire enterprise network. Specifically, avoid deleting Enterprise Replication objects such as replicates, replicate sets, and Enterprise Replication servers, and immediately recreating 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 recreate a deleted 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 was successfully propagated through the entire Enterprise Replication system before you recreate the object.

Backup and Restore Considerations

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

Warm restores are not permitted. You must perform a cold restore up to the current log of all relevant dbspaces on Enterprise Replication servers before resuming replication.

If the restore did not include all the log files from the replay position, or the system was not restored to the current log file, you must advance the log file unique ID past the latest log file unique ID prior to the restore, and then run the **cdr cleanstart** command followed by the **cdr sync** command to synchronize the server.

Data Compression Considerations

You can compress and uncompress data in replicated tables to reduce the amount of needed disk space.

You can also consolidate free space in a table or fragment and you can return this free space to the dbspace. Performing these operations on one Enterprise Replication server does not affect the data on any other Enterprise Replication server.

Attention: After you uncompress data on one server, do not remove any compression dictionaries that another Enterprise Replication server needs.

For more information on compression operations, see the *IBM Informix Administrator's Guide*.

Database and Table Design Considerations

These topics describe how to design databases and tables for replication.

Unbuffered Logging

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

It is recommended 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, IBM Informix 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
```

To minimize impact on the system, IBM Informix Enterprise Replication uses buffered logging whenever possible, even if the database is defined as unbuffered. For more information, see the section on CREATE DATABASE in the *IBM Informix Database Design and Implementation Guide*.

Table Types

Enterprise Replication has restrictions on the types of tables that can participate in replication.

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, do not include these tables in your replication environment.

Enterprise Replication imposes the following operational limitations:

- 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 about views and synonyms, see the *IBM Informix Database Design and Implementation Guide*.
- Replication is not inherited by any child tables in a typed hierarchy.

For more information about table types, see *IBM Informix Database Design and Implementation Guide*.

Label-based access control

You cannot apply label-based access control (LBAC) to a table participating in Enterprise Replication. Nor can you define an Enterprise Replication replicate on a table that is protected by LBAC.

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, unless the data is a smart large object. Therefore, you should avoid placing a NOT NULL constraint on any replicated column that includes out-of-row data.

If a column has smart large objects and the smart large object data does not exist when Enterprise Replication collects it for replication, then Enterprise Replication creates smart large objects with no data and zero size.

Shadow Columns

Shadow columns are hidden columns on replicated tables that contain values supplied by the database server. The database server uses shadow columns to perform internal operations.

You can add shadow columns to your replicated tables with the CREATE TABLE or ALTER TABLE statement. To view the contents of shadow columns, you must

explicitly specify the columns in the projection list of a SELECT statement; shadow columns are not included in the results of SELECT * statements.

The CRCOLS shadow columns, **cdrserver** and **cdrtime**, support conflict resolution. These two columns are hidden shadow columns because they cannot be indexed and cannot be viewed in the system catalog tables. In an update-anywhere replication environment, you must provide for conflict resolution using a conflict resolution rule. When you create a table that uses the time stamp, time stamp plus SPL, or delete wins conflict resolution rule, you must define the shadow columns, **cdrserver** and **cdrtime** on both the source and target replication servers. If you plan to use only the ignore or always-apply conflict resolution rule, you do not need to define the **cdrserver** and **cdrtime** shadow columns for conflict resolution.

The REPLCHECK shadow column, **ifx_replcheck**, supports faster consistency checking. This column is a visible shadow column because it can be indexed and can be viewed in the system catalog table. If you want to improve the performance of the **cdr check replicate** or **cdr check replicateset** commands, you can add the **ifx_replcheck** shadow column to the replicate table, and then create a new index that includes the **ifx_replcheck** shadow column and your primary key columns.

The ERKEY shadow columns, **ifx_erkey1**, **ifx_erkey2**, **ifx_erkey3**, are used in place of a primary key on replicated tables. If you create replicated tables through a grid, the ERKEY columns are automatically added.

Related concepts:

"Conflict Resolution" on page 3-6

"Preparing Tables for Conflict Resolution" on page 4-19

"Shadow Column Disk Space" on page 4-9

"Preparing Tables for a Consistency Check Index" on page 4-20

"Load and unload data" on page 4-24

Primary Key Constraint

All tables involved in replication must have a PRIMARY KEY constraint defined on at least one column or have the ERKEY shadow columns defined. The primary key must be the same on all servers that participate in the replicate.

The ERKEY shadow columns provide Enterprise Replication with a unique key to use as a primary key for replication purposes.

For more information about primary keys and constraints, see the *IBM Informix Database Design and Implementation Guide* and the *IBM Informix Guide to SQL: Syntax*.

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

Related tasks:

"Preparing tables without primary keys" on page 4-21

Serial Data Types and Primary Keys

If you plan to use serial data types (SERIAL, SERIAL8, or BIGSERIAL) 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 Configuration Parameter" on page B-13.

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.

Related tasks:

"Setting Configuration Parameters" on page 4-14

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 deletions on the parent tables first and then the children tables. When Enterprise Replication processes deletions on the children, an error might result, because the rows were already deleted when the parent was deleted. The table in Table 2-1 indicates how IBM Informix 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 *IBM Informix Guide to SQL: Syntax*.

Table 2-1. Resolving Cascade Deletes

Conflict-Resolution Rule	Conflict-Resolution Scope	Actions on Delete Errors
Time stamp	Row-by-row or transaction	Continue processing rest of the transaction
Delete wins	Row-by-row or transaction	Continue processing rest of the transaction
Ignore	Transaction	Abort entire transaction
Ignore	Row-by-row	Continue processing rest of the transaction

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 execute 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 Table 1-2 on page 1-10), 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.

You might want to design your triggers to take different actions depending on whether a transaction is being performed as part of Enterprise Replication. Use the **'cdrsession'** option of the **DBINFO()** function to determine if the transaction is a replicated transaction. The **DBINFO('cdrsession')** function returns 1 if the thread performing the database operation is an Enterprise Replication apply or sync thread; otherwise, the function returns `0`.

For more information on triggers, see "Enabling Triggers" on page 6-13 and the CREATE TRIGGER section in *IBM Informix Guide to SQL: Syntax*.

Related reference:

DBINFO Function (SQL Syntax)

Using Constraints

When using constraints, ensure that the constraints you add at the target server are not more restrictive than those at the source server. Discrepancies between constraints at the source and target servers can cause some rows to fail to be replicated.

For tables that have referential integrity constraints set up between them, if you need to resynchronize the data in the tables, you can perform synchronization on the replicate set. For replicate sets, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables).

When you perform synchronization, rows that fail to be repaired due to discrepancies between constraints are recorded in the ATS and RIS files. For more information about ATS and RIS files, see Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1.

Sequence Objects

In bi-directional Enterprise Replication, if you replicate tables using sequence objects for update, insert, or delete operations, the same sequence values might be generated on different servers at the same time, leading to conflicts.

To avoid this problem, define sequence objects on each server so that the ranges of generated sequence values are distinct. For more information about the CREATE SEQUENCE and ALTER SEQUENCE statements of SQL, see the *IBM Informix Guide to SQL: Syntax*.

The NLSCASE database property

Enterprise Replication supports both case-sensitive databases and NLSCASE INSENSITIVE databases. (Databases created with the NLSCASE INSENSITIVE option ignore letter case in operations on NCHAR and NVARCHAR strings, and on strings of other character data types that are cast explicitly or implicitly to NCHAR or NVARCHAR data types.)

The database server does not prevent a case-sensitive database from being replicated by a database that has the NLSCASE INSENSITIVE property, nor the

replication of an NLSCASE INSENSITIVE database by a case-sensitive database. No warning or exception is issued by the database server in either of these cases when you define replication participants.

These two types of database behave differently, however, in operations that classify NCHAR and NVARCHAR strings as duplicates or as distinct values, if the character strings that are being compared differ only in letter case. It is the user's responsibility to make sure that replication participants with different NLSCASE attributes will not cause exceptions or unexpected behavior when replicating the results of operations like the following on NCHAR or NVARCHAR data:

- sorting and collation
- foreign key and primary key dependencies
- enforcing unique constraints
- clustered indexes
- access-method optimizer directives
- queries with WHERE predicates
- queries with UNIQUE or DISTINCT specifications in the projection clause
- queries with ORDER BY clauses
- queries with GROUP BY clauses
- cascading DELETE operations
- table or index storage fragmentation BY EXPRESSION
- table or index storage fragmentation BY LIST
- data distributions from UPDATE STATISTICS operations

To avoid the risk of consistency problems that can result from differences in case-sensitivity, the following policy might be useful when you define replication pairs:

- Replicate case-sensitive databases only with case-sensitive databases.
- Replicate NLSCASE INSENSITIVE databases only with NLSCASE INSENSITIVE databases.

Related concepts:

Duplicate rows in NLSCASE INSENSITIVE databases (SQL Syntax)

Transaction Processing Considerations

Many variables affect what impact replicating data has on your transaction processing.

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, Informix 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 *IBM Informix Administrator's Guide*.

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 the application of one transaction and another (depending on how much transaction data is in each server's send queue and the state of the server).

Large Transactions

While Enterprise Replication is able to handle large transactions, it is optimized for small transactions. For best performance, avoid replicating large transactions.

Large transactions are handled with a grouper paging file located in temporary smart large objects. Enterprise Replication can process transactions up to 4 TB in size. For more information, see "Setting Up the Grouper Paging File" on page 4-13. You can view Enterprise Replication grouper paging statistics with the **onstat -g grp pager** command (see "onstat -g grp" on page E-8).

Instead of using Enterprise Replication to perform a batch job, use BEGIN WORK WITHOUT REPLICATION to run the batch job locally on each database server. For more information, see "Blocking Replication" on page 4-17.

Supported SQL Statements

After you define Enterprise Replication on a table by including that table as a participant in a replicate 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.

To use the forbidden and limited SQL statements described in this section 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 at all required nodes, restart the replicate. For more information, see "Managing Replicates" on page 8-6.

Forbidden SQL Statements: You cannot use the following SQL statement against a table that is included in a replicate:

- DROP TABLE

Limited SQL Statements:

You cannot execute some SQL statements when replication is active. To use these SQL statements, you must first put the replicated table in alter mode or stop replication.

You must first set alter mode with the **cdr alter** command before you can perform these tasks:

- Add shadow columns:
  - ALTER TABLE ... ADD CRCOLS;
  - ALTER TABLE ... ADD REPLCHECK;
  - ALTER TABLE ... ADD ERKEY
- Remove or disable the primary key constraint.
- Modify the primary key columns.

For example, alter the column to add default values or other integrity constraints.

- Change the primary key from one column to another.

  For example, if a primary key is defined on **col1**, you can change the primary key to **col2**.

You must stop replication before performing these tasks:

- Drop conflict resolution shadow columns with ALTER TABLE ... DROP CRCOLS.
- Add or drop rowids.

Related concepts:

"Preparing Tables for a Consistency Check Index" on page 4-20

"Preparing Tables for Conflict Resolution" on page 4-19

Related tasks:

"Considerations for Changing or Recreating Primary Key Columns" on page 8-26

"Preparing tables without primary keys" on page 4-21

Related reference:

"cdr alter" on page A-29


Enterprise Replication shadow columns (SQL Syntax)

Permitted SQL Statements: IBM Informix Enterprise Replication permits the following SQL statements with no limitations:

- ADD INDEX
- ALTER INDEX . . . TO CLUSTER
- ALTER FRAGMENT
- ALTER INDEX
- ALTER TABLE (except for the primary key)
- CREATE CLUSTER INDEX
- CREATE SYNONYM
- CREATE TRIGGER
- CREATE VIEW
- DROP INDEX
- DROP SYNONYM
- DROP TRIGGER
- DROP VIEW
- RENAME COLUMN
- RENAME DATABASE
- RENAME TABLE
- SET *object mode* (no disabling of primary key constraint)
- START VIOLATIONS TABLE
- STOP VIOLATIONS TABLE
- TRUNCATE TABLE

Tip: You can rename both dbspaces and sbspaces while IBM Informix Enterprise Replication is active.

For more information on requirements for SQL statements see “Alter, Rename, or Truncate Operations during Replication” on page 8-22.

Using GLS with Enterprise Replication

You can replicate using non-default locales.

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 U. S. English locale.

  In other words, the **syscdr** database on every Enterprise Replication server must be in the English 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 Connect 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 *IBM Informix GLS User's Guide*.

Related concepts:

“Flexible Architecture” on page 1-4

Related tasks:

“Enabling code set conversion between replicates” on page 6-14

Using Enterprise Replication in Mixed-Version Environments

You can set up Enterprise Replication across servers of different version levels. Enterprise Replication stores an internal version number that it communicates to other servers on initiating a connection with them. Each Enterprise Replication server instance can only use the features supported by its version level. Attempts to use features from later releases with previous versions of Enterprise Replication raise errors.

Enterprise Replication data types

Enterprise Replication supports built-in data types and user-defined data types, including row types and collection types.

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

Important: For non-master replicates, 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, SERIAL8, or BIGSERIAL data types, you must be careful when defining serial columns. For more information, see “Serial Data Types and Primary Keys” on page 2-7.

Related concepts:

"Flexible Architecture" on page 1-4

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 6-13.

Replicating Simple and Smart Large Objects

How Enterprise Replication handles simple and smart large objects depends on how the objects are stored.

Enterprise Replication replicates the following types of large objects:

- Simple large object data types (TEXT and BYTE)

  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 (BLOB and CLOB)

  You must store smart large objects in sbspaces.

Simple large objects in tblspaces are logged in the logical log and therefore, Enterprise Replication can evaluate the data for replication directly. Enterprise Replication cannot evaluate large object data that is stored in a blobspace or sbspace; instead, Enterprise Replication uses information about the large object that is stored in the row to evaluate them.

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

Replicating Simple Large Objects from Tblspaces:

Enterprise Replication evaluates simple large object data that is stored in a tblspace independently from the rest of its row.

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.

By default, Enterprise Replication performs time stamp and delete wins 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.

For simple large objects, if the column on the target database server is also stored in a tblspace, Enterprise Replication evaluates the values of the shadow columns, **cdrserver** and **cdrtime**, 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.
- If there is no SPL conflict-resolution rule and the time stamps are equal, then Enterprise Replication applies the data to the row with the lowest CDR server ID.

If you use the SPL conflict resolution, simple large objects that are stored in tblspaces are handled differently than large objects in blobspaces.

Related concepts:

"SPL Conflict Resolution for Large Objects" on page 3-11

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 or sbspace before sending the data to the target database server.

It is possible that a transaction subsequent to the transaction being replicated can modify or delete 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 for only a short 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 6-12.

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.

If you use the SPL conflict resolution, large objects that are stored in blobspaces or sbspaces are handled differently than simple large objects in tblspaces.

Related concepts:

"SPL Conflict Resolution for Large Objects" on page 3-11

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 *IBM 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 data types and extended data types, including opaque data types and user-defined types (UDTs).

Installing and Registering UDTs: You must install and register UDTs and their associated support routines on all database servers participating in Enterprise Replication prior to starting replication.

If you combine Enterprise Replication with high-availability clusters, you must install UDTs on both the primary and secondary database servers, but only register them on the primary database server (see *IBM Informix Administrator's Guide*).

UDT Support Functions: If you plan to replicate opaque user-defined types (UDTs), the UDT designer must provide the following types of support functions:

- **streamwrite()** and **streamread()** functions
- **compare()** and its supporting **greaterthan()**, **lessthan()**, and **equal()** functions

This also applies to UDTs embedded in complex types.

The streamread() and streamwrite() Functions

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 *IBM Informix DataBlade API Programmer's Guide*.

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.

Comparison Functions

Enterprise Replication uses comparison functions to determine if a replicated column has been altered. For example, the comparison functions are used when the replicate definition specifies to replicate changed columns only with the **–fullrow n** option.

When you define a **compare()** function, you must also define the **greaterthan()**, **lessthan()**, **equal()**, or other functions that use the **compare()** function.

For more information on writing these support functions, see the *IBM Informix User-Defined Routines and Data Types Developer's Guide*.

Considerations for Replicating Opaque Data Types:

Opaque data types can be replicated, but have certain restrictions.

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).
- 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.
- 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 *IBM Informix Guide to SQL: Syntax*.

Related concepts:

"Conflict Resolution" on page 3-6

Related tasks:

"Replicating Only Changed Columns" on page 6-12

Related reference:

"Participant and participant modifier" on page A-4

"cdr start sec2er" on page A-137

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.

Verifying the Data Type of Replicated Columns

By using master replicates you can verify that all participants in a replicate have columns with matching data types. Master replicates also allow verification that each participant contains all replicated columns, and optionally that column names are the same on each participant. See "Defining Master Replicates" on page 6-8 for more information.

Part 2. Setting Up and Managing Enterprise Replication

These topics describe setting up Enterprise Replication by designing the replication system, preparing the environment, instantiating the replication system, and then maintaining it.

Chapter 3. Selecting the Enterprise Replication System and Network Topology

These topics describe types of replication systems provided by Enterprise Replication and discuss the trade-offs associated with performance and data availability.

Primary-Target Replication System

In the primary-target replication system, the flow of information is in one direction.

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 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.

Related concepts:

"Flexible Architecture" on page 1-4

"Participant definitions" on page 6-7

Related reference:

"Participant and participant modifier" on page A-4

Primary-Target Data Dissemination

Data dissemination supports business needs where data is updated in a central location and then replicated to servers that only receive data and do not update data.

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 slow the performance of the application. The following figure illustrates data dissemination.

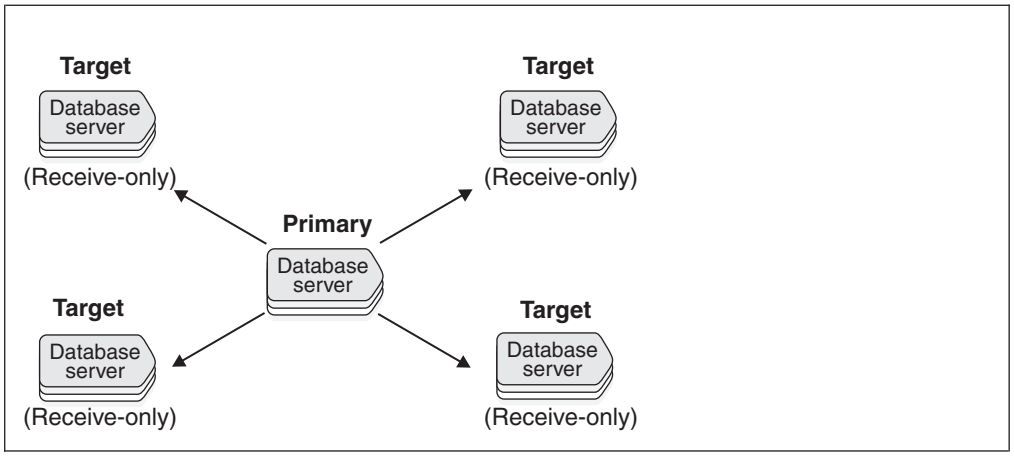


Figure 3-1. Data Dissemination in a Primary-Target Replication System

Data consolidation

Businesses can choose to consolidate data into one or more central database servers.

Data consolidation allows the migration of data from several database servers to a central database server. For example, several retail stores can replicate inventory and sales information to headquarters. The retail stores do not need information from other stores but headquarters needs the total inventory and sales of all stores.

In the following figure, the remote locations only send data while a single central database server only receives data.

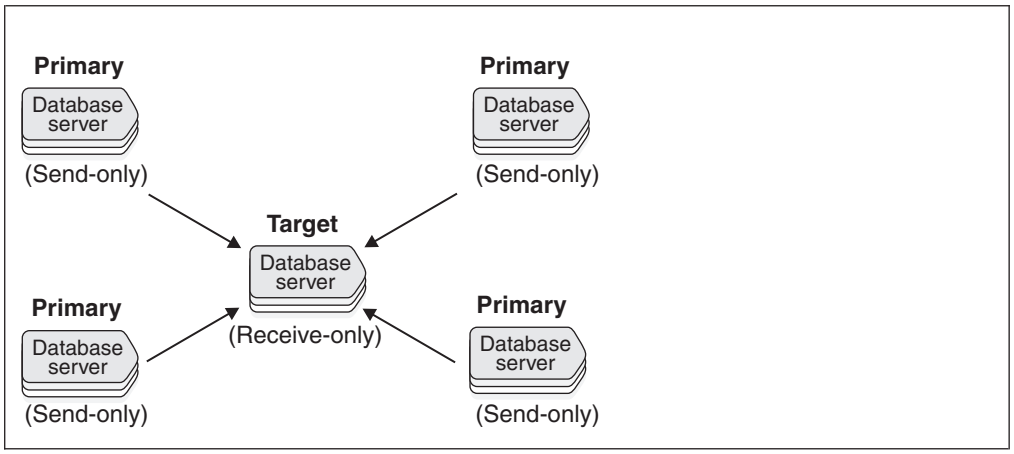


Figure 3-2. Data Consolidation in a Primary-Target Replication System

You can also use data consolidation to replicate data from many database servers to more than one central database server. For example, a retail chain has two central database servers, one for the eastern half of the United States, and one for the western half of the United States. The retail stores replicate data to their designated central server and the two central servers replicate data to each other. In this configuration, the replication servers in the retail stores only send data to the central servers, but the central servers both send and receive data.

Businesses can use data consolidation to replicate OLTP data to a dedicated computer for decision support (DSS) analysis. For example, data from several OLTP systems can be replicated to a DSS system for read-only analysis.

The primary key for every replicated row must be 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.

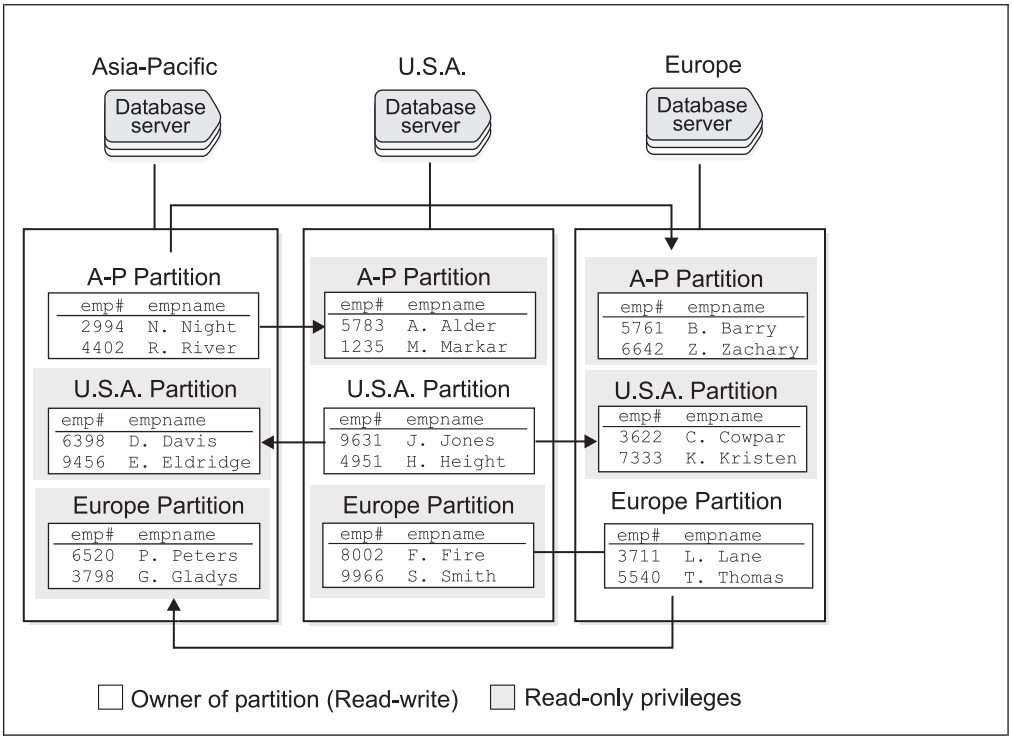


Figure 3-3. Workload Partitioning in a Primary-Target Replication System

In Figure 3-3, the replication model matches the partition model for the **employee** tables. The Asia-Pacific database server 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. The Asia-Pacific database server 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.

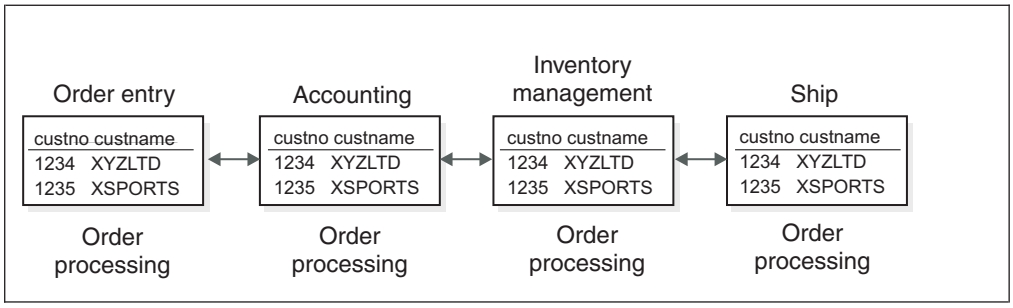


Figure 3-4. A Workflow-Replication System Where Update Authority Moves From Site to Site

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

Consider the following factors 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 3-6.

- Capacity planning

  All replication systems require you to plan for capacity changes. For more information, see "Preparing Data for Replication" on page 4-17.

- 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 a high-availability replication system. For more information, see "High-Availability Replication System" on page 5-1.

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.

The following figure illustrates an update-anywhere replication system where the service centers in Washington, New York, and Los Angeles each replicate changes to the other two servers.

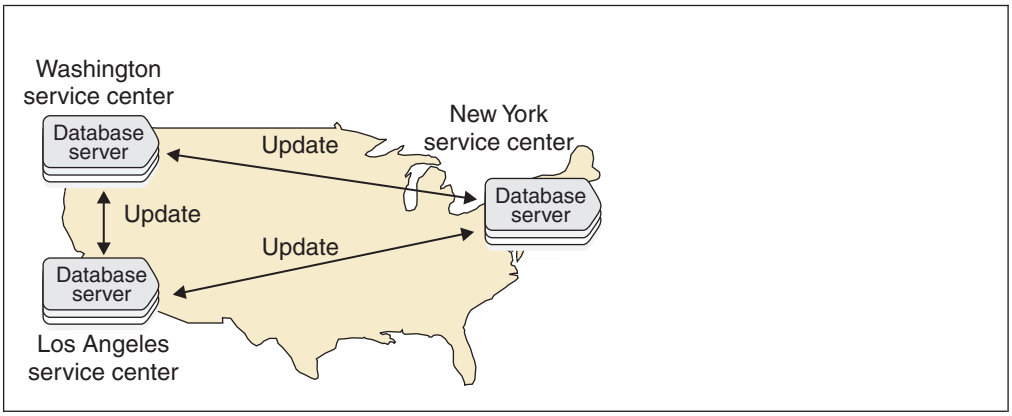


Figure 3-5. Update-Anywhere Replication System

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.

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. 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 for resolving 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) outweigh your need for a fail-safe mechanism to deliver consistent information.

- Capacity Planning

  All replication systems require you to plan for capacity changes and prepare the data for replication. If you choose a time-based conflict resolution rule, you need to provide space for delete tables and add shadow columns to replicated tables.

- High Availability

  If any of your database servers are critical, consider using high-availability clusters to provide backup servers.

Related concepts:

"High-Availability Replication System" on page 5-1
"Disk Space for Delete Tables" on page 4-8
"Shadow Column Disk Space" on page 4-9
"Preparing Data for Replication" on page 4-17
"Flexible Architecture" on page 1-4

Related tasks:

"Specifying Conflict Resolution Rules and Scope" on page 6-10

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 occurs. For more information, see "Applying Replicated Data" on page 1-12. Enterprise Replication must determine which new data to replicate. To solve conflict resolution, you must specify the following for each replicate:

- A conflict-resolution rule
- The scope of the rule

Related concepts:

"Shadow Columns" on page 2-6
"Time Synchronization" on page 4-16
"Considerations for Replicating Opaque Data Types" on page 2-17

Related tasks:

"Replicating Only Changed Columns" on page 6-12

Related reference:

"cdr define replicate" on page A-61

Conflict Resolution Rule

The conflict resolution rule determines how conflicts between replicated transactions are resolved.

Enterprise Replication supports the following conflict resolution rules.

Conflict Resolution Rule	Effect
Ignore	Enterprise Replication does not attempt to resolve conflicts.
Time stamp	The row or transaction with the most recent time stamp is applied.
SPL routine	Enterprise Replication uses a routine written in SPL (Stored Procedure Language) that you provide to determine which data should be applied.
Time stamp with SPL routine	If the time stamps are identical, Enterprise Replication invokes an SPL routine that you provide to resolve the conflict.
Delete wins	DELETE and INSERT operations win over UPDATE operations; otherwise the row or transaction with the most recent time stamp is applied.
Always-apply	Enterprise Replication does not attempt to resolve conflicts.

Related tasks:

"Specifying Conflict Resolution Rules and Scope" on page 6-10

"Creating replicated tables through a grid" on page 7-10

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. Use the ignore conflict-resolution rule only with a primary-target replication system. If you use ignore with an update-anywhere replication system, your data might become inconsistent.

The ignore conflict-resolution rule can be used only as a primary conflict-resolution rule and can have either a transaction or a row scope (as described in "Conflict Resolution Scope" on page 3-14).

The following table describes how the ignore conflict resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 3-1. Ignore Conflict-Resolution Rule

Row Exists in Target?	INSERT	UPDATE	DELETE
No	Apply row	Discard row	Discard row
Yes	Discard row	Apply row	Apply row

When a replication message fails to apply to 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 cannot be applied to a target is written to this directory.

For more information, see Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1.

Time Stamp Conflict Resolution Rule

The time stamp rule evaluates the latest time stamp of the replication against the target and determines how to resolve any conflict.

All time stamps and internal computations are performed in Greenwich mean time (GMT). The time stamp conflict resolution rule assumes time synchronization between cooperating Enterprise Replication servers.

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.

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

If no secondary conflict-resolution rule is defined and the time stamps are equal, the transaction from the database server with the lower value in the **cdrserver** shadow column wins the conflict.

The following table 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.

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

The following table describes how the time stamp conflict resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 3-2. Conflict Resolution Based on the Time Stamp

Row Exists on Target?	Time Stamp	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
Yes	$T_{last\_update} > T_{repl}$	Discard row	Discard row	Discard row
Yes	$T_{last\_update} = T_{repl}$	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.	Apply row if no routine is defined as a secondary conflict resolution rule. Otherwise, run the routine.

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.

To use time stamp conflict resolution while repairing inconsistencies with the **cdr check replicate** or **cdr check replicateset** command, include the **--timestamp** option with the **--repair** option. If you need to temporarily stop replication on a server whose replicates use the time stamp conflict resolution rule, disable the

replication server with the **cdr disable server** command. When you disable a server, information about deleted rows is kept in the delete tables to be used during the time stamp repair after the server is enabled.

Related concepts:

"Conflict Resolution Scope" on page 3-14

"Time Synchronization" on page 4-16

"Delete Wins Conflict Resolution Rule" on page 3-12

"Repair inconsistencies by time stamp" on page 8-20

Related reference:

"cdr disable server" on page A-86

SPL Conflict Resolution Rule

You can write an SPL routine as a primary conflict resolution rule or as secondary conflict resolution rule to the time stamp conflict resolution rule.

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.

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 an SPL routine or a time stamp with an 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 6-12.

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)] or Local key delete-row indicator [CHAR(1)]	Y indicates the origin of the row is the delete table. K indicates the origin of the row is the replicate-key delete row. If a conflict occurs while deleting a primary key row, because the local row with the old key no longer exists, the received key delete row is passed as the local row (using the seventh argument, local row data, described below). The received key insert row is passed to the stored procedure as the replicated row using the eighth argument, described below.
Server name [CHAR(18)]	Of the replicate source
Time stamp (DATETIME YEAR TO SECOND)	From the replicated row

Argument	Description
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 an SQL error. • O - Discard the replicated row. • X - Abort the transaction.
A non-zero integer value to request logging of the conflict 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 routine returns one of the following replicate action types: 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-63.

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

SPL Conflict Resolution for Large Objects:

If the replicate is defined with an SPL conflict-resolution rule, the SPL routine must return the desired action for each smart large object (BLOB or CLOB) and simple large object (BYTE or TEXT) column.

When the routine is invoked, information about each large object 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 that the large object data is sent from the source database server. • `U` indicates that the large object data is unchanged on the source database server.
Column type [CHAR(1)]	• `P` indicates tblspace data. • `B` indicates blobspace data. • `S` indicates sbspace data.
ID of last update server [CHAR(18)]	The ID of the database server that last updated this column for tblspace data. For blobspace data: NULL For sbspace data: NULL
Last update time (DATETIME YEAR TO SECOND)	For tblspace data: The date and time when the data was last updated. For blobspace data: NULL For sbspace data: NULL

If the routine returns an action code of `A`, `D`, `I`, or `U`, the routine parses the return values of the replicated columns. Each large object column can return a two-character field.

The first character defines the desired option for the large object 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 or sbspace 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.

Related concepts:

"Replicating Simple Large Objects from Tblspaces" on page 2-14

"Replicating Large Objects from Blobspaces or Sbspaces" on page 2-15

Delete Wins Conflict Resolution Rule

The delete wins rule ensures that DELETE and INSERT operations win over UPDATE operations and that all other conflicts are resolved by comparing time stamps.

All time stamps and internal computations are performed in Greenwich mean time (GMT). The delete wins conflict-resolution rule assumes time synchronization between cooperating Enterprise Replication servers.

The delete wins rule is similar to the time stamp rule except that it prevents upsert operations and does not use a secondary conflict resolution rule. The delete wins rule prevents *upsert* operations that results from an UPDATE operation that is converted to an INSERT operation because the row to update was not found on the target server. An upsert operation can occur if a row is deleted from a target server before an UPDATE operation is processed on that target server or if an UPDATE operation was processed by the target server before the INSERT operation for that row. Depending on your business logic, upsert operations might violate referential integrity.

The delete wins rule prevents upsert operations in the following ways:

- If a row is deleted on a replication server, that row is deleted on all other replication servers, regardless of whether an UPDATE operation to that row occurred after the delete.
- If an UPDATE operation to a row is received before its INSERT operation, the UPDATE operation fails and generates and ATS or RIS file. The INSERT operation succeeds, but results in data inconsistency. To repair the inconsistency, run the **cdr check replicate** command with the **--repair** option.

The delete wins rule handles time stamp conflicts 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.

- 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 the time stamps are equal, the transaction from the database server with the lower value in the **cdrserver** shadow column wins the conflict.

The table below shows how a conflict is resolved with the delete wins rule 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. If the row exists, Enterprise Replication uses the latest time stamp to resolve the conflict.

The following table describes how the delete wins conflict resolution rule handles INSERT, UPDATE, and DELETE operations that are performed on the source server.

Table 3-3. Conflict Resolution Based on the Time Stamp

Row Exists on Target?	**Time Stamp**	**INSERT**	**UPDATE**	**DELETE**
No	n/a	Apply row	Discard row and generate and ATS or RIS file	Apply row (INSERT into Enterprise Replication delete table)
Yes	$T_{last\_update} < T_{repl}$	Apply row (Convert INSERT to UPDATE)	Apply row	Apply row
Yes	$T_{last\_update} > T_{repl}$	Discard row	Discard row	Apply row
Yes	$T_{last\_update} = T_{repl}$	The server with the lower value in the **cdrserver** shadow column wins the conflict.	The server with the lower value in the **cdrserver** shadow column wins the conflict.	The server with the lower value in the **cdrserver** shadow column wins the conflict.

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.

To use delete wins conflict resolution while repairing inconsistencies with the **cdr check replicate** or **cdr check replicateset** command, include the **--timestamp** and **--deletewins** options with the **--repair** option. Also set the CDR_DELAY_PURGE_DTC configuration parameter to the maximum age of modifications to rows that are being actively updated. If you need to temporarily stop replication on a server whose replicates use the delete wins conflict resolution rule, disable the replication server with the **cdr disable server** command. When you disable a server, information about deleted rows is kept in the delete tables to be used during the time stamp repair after the server is enabled.

Related concepts:

"Time Synchronization" on page 4-16

"Time Stamp Conflict Resolution Rule" on page 3-7

"Repair inconsistencies by time stamp" on page 8-20

Related reference:

"cdr disable server" on page A-86

"CDR_DELAY_PURGE_DTC configuration parameter" on page B-2

Always-Apply Conflict-Resolution Rule

The always-apply conflict-resolution rule does not attempt to detect or resolve conflicts.

Unlike with the ignore conflict-resolution rule, replicated changes are applied even if the operations are not the same on the source and target servers. If a conflict occurs, the current row on the target is deleted and replaced with the replicated row from the source. Use the always-apply conflict-resolution rule only with a primary-target replication system. If you use always-apply with an update-anywhere replication system, your data might become inconsistent.

The following table describes how the always-apply conflict-resolution rule handles INSERT, UPDATE, and DELETE operations.

Table 3-4. Always-Apply Conflict-Resolution Rule

Row exists in target?	INSERT	UPDATE	DELETE
No	Apply row	Apply row (convert UPDATE to INSERT)	Apply row (no error returned)
Yes	Apply as an UPDATE (overwrite the existing row)	Apply row	Deletes the row

Conflict Resolution 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.

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.

Transaction and row scopes are only applicable for apply failure related to conflict resolution, such as missing rows or newer local rows. For other errors, such as lock timeouts, constraint problems, lack of disk space, and so on, the whole incoming transaction is aborted, rolled back, and spooled to ATS or RIS files if so configured, regardless of whether row scope is defined.

Related concepts:

"Failed Transaction (ATS and RIS) Files" on page 9-3

"Time Stamp Conflict Resolution Rule" on page 3-7

Related tasks:

"Specifying Conflict Resolution Rules and Scope" on page 6-10

Related reference:

"cdr define replicate" on page A-61

Choosing a Replication Network Topology

Enterprise replication *topology* describes connections that replication servers make to interact with each other. This topology is the route of replication data (message) transfer from server to server over the network. The replication topology is not synonymous with the physical network topology. Replication server definitions create the replication topology, whereas replicate definitions determine data to be replicated and the sources and destinations within the topology.

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

Related concepts:

"Flexible Architecture" on page 1-4

Related tasks:

"Defining Replication Servers" on page 6-1

"Customizing the Replication Server Definition" on page 6-6

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 3-6 on page 3-16 shows a fully connected replication topology. Each database server connects directly to every other database server in the replication environment.

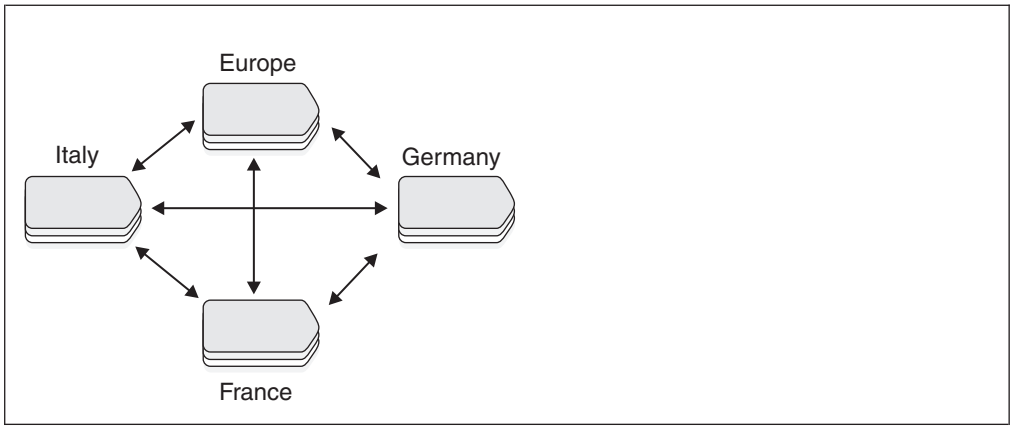


Figure 3-6. Fully Connected Topology

If necessary, you can also add high-availability clusters and a backup server to any server to provide high availability. For more information, see "High-Availability Replication System" on page 5-1.

Hierarchical Routing Topology Terminology

Enterprise Replication uses the terms in the Table 3-5 to describe Hierarchical Routing topology.

Table 3-5. 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 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 3-6 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 *Hierarchical Routing* 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 6-6 and "cdr define server" on page A-71.

Related concepts:

"Database Server Groups" on page 4-2

Related reference:

"The syscdrs Table" on page G-14

Hierarchical Tree Topology

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 3-7 illustrates a replication tree.

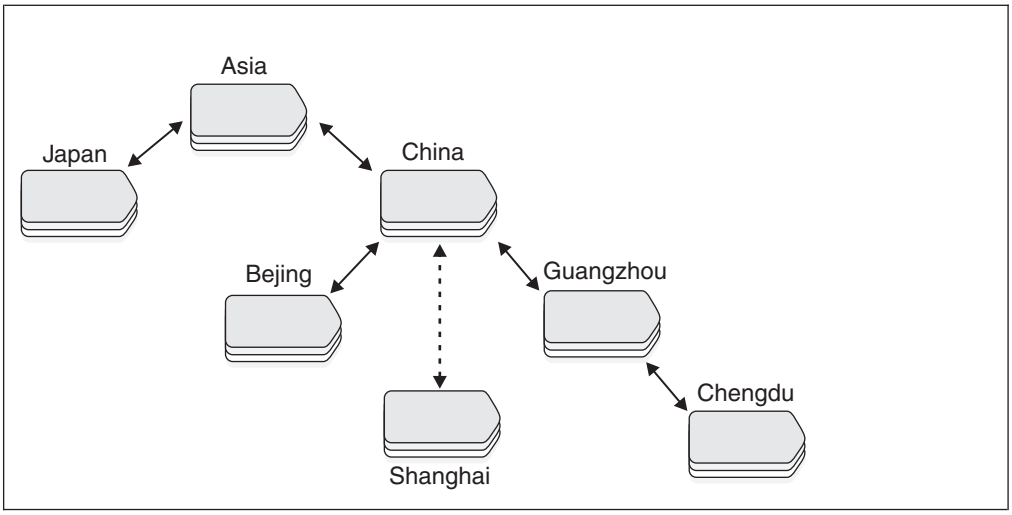


Figure 3-7. Hierarchical Tree Topology

In Figure 3-7, 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 **Chengdu**.

Asia is the root database server. **Japan**, **China**, and **Guangzhou** are nonroot database servers. You can define **Beijing**, **Shanghai**, and **Chengdu** 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.

You can define a replicate that replicates data exclusively between **Shanghai** and **Japan**. However, the transaction data would must go through **China** and **Asia**. If either **China** or **Asia** is offline replication is suspended. Similarly, a replicate defined between **Japan** and **China** would require **Asia** to be functioning, even though both **Japan** and **China**, as nonroot servers, have entries in their `sqlhosts` files for each other.

Parent servers are good candidates for using high-availability clusters to provide backup servers.

Forest of trees topology

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 3-8 shows a forest of trees.

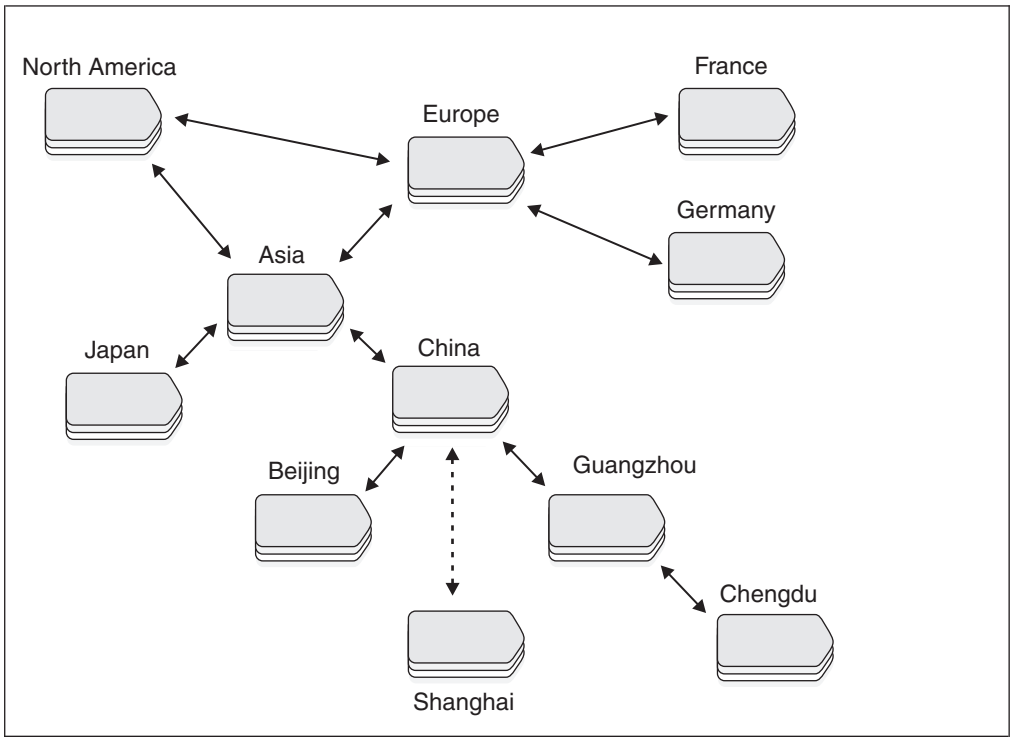


Figure 3-8. Forest-of-Trees Topology

In Figure 3-8, **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 3-8 were fully connected, instead of being organized in trees, 55 connections would be required.

To ensure that all servers retain access to the replication system, use high-availability clusters on parent servers. For more information, see "Using high-availability clusters in a forest of trees topology" on page 5-4.

Chapter 4. Preparing the Replication Environment

These topics cover the steps to take to prepare your environment for replicating data with Enterprise Replication: preparing the network environment, the disk, the server environment, and the data.

Related tasks:

"Creating a new domain by cloning a server" on page 6-2

Preparing the Network Environment

You must prepare the network environment for each database server that will be involved in Enterprise Replication.

The following files are involved in configuring the replication network:

- `sqlhosts`: Specifies replication connectivity, including server groups, connection security, and network security.
- `hosts`: Specifies hosts names if you are not using Domain Name Service (DNS).
- `services`: Specifies the service name associated with a port number.
- `hosts.equiv`: Specifies trusted host names of replication servers.

For more information on preparing the network environment, see the chapter on client/server connectivity in the *IBM Informix Administrator's Guide*.

To prepare your network environment:

1. If you are not using DNS, configure replication server host information on each replication server in the `hosts` files.
2. Configure port information in the `services` files and the `sqlhosts` files.
3. Create database server groups for all replications servers in the `sqlhosts` files.
4. If necessary, configure secure connections for replication servers in the `sqlhosts` files and the `hosts.equiv` files.
5. If necessary, configure network security for client/server communications in the `sqlhosts` files.
6. Test the replication network, if you are not configuring secure connections in the `sqlhosts` files.

Related concepts:

Appendix H, "Replication Examples," on page H-1

Configuring hosts information for replication servers

If you are not using Domain Name Service (DNS) to identify IP addresses and system names, you do need to configure the `hosts` file on each replication server to add the IP addresses and system names for all other replication servers in the domain.

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:

```
192.168.0.1 ny.usa.com
192.168.0.2 tokyo.japan.com
192.168.0.3 rome.italy.com
192.168.0.4 perth.australia.com
```

Configuring port and service names for replication servers

Replication servers must know the port numbers for each of the other replication servers in the domain.

Configure port numbers for replication servers in one of the following ways:

- Specify the port numbers in the sqlhosts file. This method risks conflicting with port numbers being used by other applications.
- Specify the service names in the sqlhosts file and specify the port numbers for each service name in the services file.

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.

Related reference:

"cdr start sec2er" on page A-137

Database Server Groups

Enterprise Replication requires that all database servers participating in replication are members of database server groups. Each server in the domain must have a unique identifier; the database server group uniquely identifies a server. The database server groups are listed in the sqlhosts file on each replication server.

Database Server Groups on UNIX

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 use the name of the *database server group* of the more familiar *database server* name (that is, the name specified by the **INFORMIXSERVER** environment variable) for all references to database servers.

The exception is the **--connect** option, which can use both server name or group name. This publication also refers to a database server group as a *server group*.

This publication 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.

Each replication server must have complete `sqlhosts` server group information for the entire domain, except leaf servers in hierarchical routing topologies. Each leaf server must have `sqlhosts` connectivity information for itself and its parent (hub).

On UNIX, a database server group is defined in the `sqlhosts` file. The following example shows a simple `sqlhosts` file for four Enterprise Replication servers, **serv1**, **serv2**, **serv3**, and **serv4** and their database server groups. The first line describes the database server group **g_serv1**, which includes the database server **serv1**. Subsequent lines describe the other servers.

```
#dbservername  nettype  hostname       servicename  options

 g_serv1       group    -                     -     i=143
 serv1         ontlitcp ny.usa.com            1230  g=g_serv1
 g_serv2       group    -                     -     i=144
 serv2         ontlitcp tokyo.japan.com       1231  g=g_serv2
 g_serv3       group    -                     -     i=145
 serv3         ontlitcp rome.italy.com        1232  g=g_serv3
 g_serv4       group    -                     -     i=146
serv4         ontlitcp perth.australia.com   1233  g=g_serv4
```

The following table describes the fields in the `sqlhosts` file.

Table 4-1. sqlhosts fields

Field name	Description
dbservername	Database server group name or database server name
nettype	Type of connection (composed of the database server product, interface type, and network protocol). Enterprise Replication cannot use shared memory connections even if the replicating servers are on same computer.
hostname	The name of the database server computer.
servicename	The service name in the `services` file or the port number.
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 domain. Must be a positive integer from 1 through 32767.

It is not necessary for the DBSERVERNAME configuration parameter to be set to a network connection; however, at least one of server names listed by the DBSERVERNAME or the DBSERVERALIASES configuration parameters must be set to a network protocol. For information about database server aliases, see the *IBM Informix Administrator's Guide*.

Database Server Groups on Windows

For information about preparing the SQLHOSTS connectivity information on Windows, see Appendix I, "SQLHOSTS Registry Key (Windows)," on page I-1.

Important: Use IBM 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.

Related concepts:

"Hierarchical Routing Topology Terminology" on page 3-16

"Managing Enterprise Replication with High-Availability Clusters" on page 5-6

"Setting Up Database Server Groups for High-Availability Cluster Servers" on page 5-5

Related tasks:

"Setting Up the Database Server Group Registry Key" on page I-3

Related reference:

"cdr define template" on page A-75

"cdr define server" on page A-71

"cdr realize template" on page A-114

"cdr sync replicate" on page A-161

"cdr sync replicateset" on page A-164

"cdr check replicate" on page A-37

"cdr check replicateset" on page A-47

Group information (Administrator's Guide)

Configuring secure connections for replication servers

You can secure connections between replication servers without making the servers fully trusted by using the connection security option in the `sqlhosts` file and creating an encrypted password file.

The secure ports listed in the `sqlhosts` files can be used only for Enterprise Replication and high-availability cluster communication. You must configure a separate port for client/server communications with the local replication servers.

To configure a trusted environment for replication:

1. In the `sqlhosts` file on each server, create a server group with two connections for the local server:
   a. Create one entry with the **s=6** option to configure communication between servers.
   b. Create one entry without the **s=6** option to configure local communication with utilities, such as the **cdr** utility.

   For example:

```
#dbservername   nettype    hostname     servicename   options
gserv1          group      -            -             i=143
serv1           ontlitcp   ny.usa.com   ertest1       g=gserv1,s=6
a_serv1         ontlitcp   ny.usa.com   ertest10
```

2. In the `sqlhosts` file on each server, add entries for each of the other servers in the domain. Use the server names associated with the **s=6** options.
3. Using a text editor, create an authorization file in `$INFORMIXDIR/etc` that includes the host names of the other replication servers in the domain, each on a separate line. Each server in the list connects to the local database server instance using the s=6 port. For example, the authorization file on the server **serv1** might be:

```
#hostname
tokyo.japan.com
tokyo

rome.italy.com
rome

perth.australia.com
perth
```

4. Set the REMOTE_SERVER_CFG configuration parameter to point to the authorization file, and set the S6_USE_REMOTE_SERVER_CFG configuration parameter to 1. For example if you named the authorization file **authfile** then specify the following in the `onconfig` file:

   ```
   REMOTE_SERVER_CFG authfile
   S6_USE_REMOTE_SERVER_CFG 1
   ```

5. Using a text editor, create and save a password file. The password file includes the host name, alternate server name, user ID, and password for each server. For example, if the user ID for server **serv1** was `informix` and the password was `informix_pw`, use the following password file entry:

   ```
   serv1  a_serv1  informix  informix_pw
   ```

6. Encrypt the password file using the **onpassword** utility. For example, if you named the text file in step 5`$INFORMIXDIR/etc/server_passwords`, and you wanted the file encrypted with a key called **access_key**, use the following command:

   ```
   onpassword -k access_key -e $INFORMIXDIR/etc/server_passwords
   ```

   The encrypted file is saved in: `$INFORMIXDIR/etc/passwd_file`.

   Important: To prevent unauthorized access to the server passwords, remove the unencrypted password file, `$INFORMIXDIR/etc/server_passwords` after creating the encrypted file.
   For more information, see The onpassword utility.

If you do not configure a password file, you must run the **cdr** utility on the local computer and specify the regular connection with the Connect option, for example:

```
cdr list server --connect=a_srv1
```

Because secure ports can be used only for replication communication, you cannot test the connections until you start replication.

Related concepts:

The sqlhosts file and the SQLHOSTS registry key (Administrator's Guide)

Related tasks:

"Testing the replication network" on page 4-6

Related reference:

S6_USE_REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

The onpassword utility (Administrator's Reference)

REMOTE_SERVER_CFG configuration parameter (Administrator's Reference)

Configuring network encryption for replication servers

You encrypt client/server network communication by specifying the ENCCSM module with the communications support module (CSM) option in the `sqlhosts`

file. You encrypt Enterprise Replication communication by setting encryption configuration parameters. The ENCRYPT_CDR configuration parameter must be set to 1 or 2 to allow encryption.

You cannot configure an Enterprise Replication connection with a CSM.

To combine client/server network encryption with Enterprise Replication encryption, configure two network connections for each database server. The configuration in the SQLHOSTS file would look like the following example.

```
gserv1        group    -              -         i=143
serv1         ontlitcp ny.usa.com     ertest1   g=gserv1
c_serv1       ontlitcp ny.usa.com     ertest10  csm=(ENCCSM)
```

In this example, **serv1** and **c_serv1** are two connection ports on the same database server. Encrypted client/server communication uses the **c_serv1** port, while encrypted Enterprise Replication uses the **serv1** port.

For more information on encrypting client/server network communications, see the *IBM Informix Administrator's Guide*.

Related tasks:

"Setting Configuration Parameters" on page 4-14

Related reference:

Appendix B, "Enterprise Replication configuration parameter and environment variable reference," on page B-1

Testing the replication network

After you set up the network environment, test the connections between the replication servers. You cannot test a connection that uses the **s=6** option in the `sqlhosts` file.

To test the network environment:

1. Verify the network connection. Use the **ping** command to test the connection between two systems. For example, from **ny.usa.com**, test the connection to **tokyo.japan.com**:

   ```
   ping tokyo.japan.com
   ```

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 **ny.usa.com**, and you are testing the connection to a database server on **tokyo.japan.com**, select **serv2** and **g_serv2**.

   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 *IBM Informix DB-Access User's Guide*.

Related tasks:

"Configuring secure connections for replication servers" on page 4-4

Testing the password file

You create and encrypt a password file to allow the CDR utility to access to a secure network environment. Use these steps to test that the encrypted password file is correctly configured.

To test the password file configuration:

Use the **cdr view state -c** *remote_server_group_name* command to verify that the password file supplies the correct password to the CDR command. For example, if your remote server group was named `g_serv2`, specify the following command:

```
cdr view state -c g_serv2
```

The state of all configured enterprise replication servers is returned. If enterprise replication is not defined, but the password file is set up correctly, the following message is returned:

```
ERROR:ER not defined on g_serv2
```

If the CDR utility is unable to connect to the server or if the following error is returned then verify that `$INFORMIXDIR/etc/passwd_file` is correctly configured.

```
25539: Invalid connection-type
```

The following is an example of command output returned when Enterprise Replication and the password file are correctly configured:

```
$  cdr view state -c g_serv2
STATE
Source   ER              Capture          Network          Apply
         State           State            State            State
------------------------------------------------------------------------
g_serv2 Active           Running          Running          Running
g_serv1 Active           Running          Running          Running

.
```

Preparing the Disk

These topics describe how to prepare your disk for Enterprise Replication.

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

Logical logs must be configured correctly for Enterprise Replication.

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-watermark) 30 percent larger than LTXHWM (long-transaction high-watermark).

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

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

The database server can add logs dynamically when Enterprise Replication approaches a potential log wrap situation if the CDR_MAX_DYNAMIC_LOGS configuration parameter is set to a non-zero integer.

Related concepts:

"Handle potential log wrapping" on page 9-15

Disk Space for Delete Tables

If you use the time stamp, time stamp and SPL routine, or delete wins conflict resolution rules, you must provide enough disk space for the *delete tables* that Enterprise Replication creates to keep track of modified rows for conflict resolution.

Delete tables handle conflicts such as when a DELETE or UPDATE operation finds no corresponding row on the target. The DTCleaner thread removes a row from the delete tables after all the servers have progressed beyond that row. Enterprise Replication does not create delete tables for tables that have replicates defined with a conflict resolution rule of ignore or always-apply.

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.

Related concepts:

"Update-Anywhere Replication System" on page 3-5

Related tasks:

"Replicating Only Changed Columns" on page 6-12

Shadow Column Disk Space

If you plan to use shadow columns, make sure to allow additional disk space for their values.

If you plan to use any conflict-resolution rule except ignore or always-apply, you must allow for an additional 8 bytes for the CRCOLS shadow columns, **cdrserver** and **cdrtime**, which store the server and time stamp information that Enterprise Replication uses for conflict resolution.

If you want to speed consistency checking by indexing the REPLCHECK shadow column, you must allow for an additional 8 bytes for the **ifx_replcheck** shadow column.

If you do not want to have a primary key on your table, or you create your replicated tables through a grid, you must allow of an additional 10 bytes for the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**. ERKEY columns also require disk space for the index that is created on them. In addition to the standard partition and page overhead, for each row in the table the ERKEY index uses 14 bytes for non-fragmented tables and 18 bytes for fragmented tables for each row in the table.

The following table shows the amount of space used by each shadow column.

Table 4-2. Shadow column size

Shadow column name	Data type	Size
cdrserver	INTEGER	4 bytes
cdrtime	INTEGER	4 bytes
ifx_replcheck	BIGINT	8 bytes
ifx_erkey_1	INTEGER	4 bytes
ifx_erkey_2	INTEGER	4 bytes
ifx_erkey_3	SMALLINT	2 bytes

The shadow columns claim disk space immediately, except when CRCOLS and ERKEY columns are added to an existing table.

Related concepts:

"Update-Anywhere Replication System" on page 3-5

"Shadow Columns" on page 2-6

"Preparing Tables for Conflict Resolution" on page 4-19

"Preparing Tables for a Consistency Check Index" on page 4-20

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 Configuration Parameter" on page B-13) 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), *replicate information* (summaries of the replication information for each transaction), and *row data* (the actual replicated data). Spooled transaction records and replication records are stored in transaction tables and replication tables in a single dbspace. Spooled row data is stored in one or more sbspaces.

Important: To prevent the send and receive queues from spooling to disk, see "Preventing Memory Queues from Overflowing" on page 9-14.

Row Data sbspaces

Replicated data might include UDT and CLOB or BLOB data types. Therefore, the spooled row data is stored as smart large objects in one or more sbspaces.

Important: Before starting Enterprise Replication, you must create at least one sbspace for spooled row data and set the CDR_QDATA_SBSPACE configuration parameter to its location.

The CDR_QDATA_SBSPACE configuration parameter accepts multiple sbspaces, up to a maximum of 32 sbspaces. Enterprise Replication can support a combination of logging and non-logging sbspaces for storing spooled row data. If CDR_QDATA_SBSPACE is configured for multiple sbspaces, then Enterprise Replication uses all appropriate sbspaces in round-robin order. For more information, see "CDR_QDATA_SBSPACE Configuration Parameter" on page B-12.

Related tasks:

"Defining Replication Servers" on page 6-1

"Setting Configuration Parameters" on page 4-14

Creating sbspaces for Spooled Row Data:

You must create dedicated sbspaces for spooled row data.

Follow these guidelines when creating sbspaces for spooled row data:

- Create all the sbspaces of same default log mode type with the same size.
- Do not use Enterprise Replication row data sbspaces for non-Enterprise Replication activity.
- Ensure that the sbspaces are sufficiently large.

  To determine the size of your spooled row data sbspaces, 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 4 MB of replicated data are logged each day, the size of the sbspaces you identify for the spooled row data must be a total of at least 4 MB.

  Windows Only

  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 differs on Windows.)

Important: When the row data sbspaces fill, Enterprise Replication hangs until you either resolve the problem that is causing Enterprise Replication to spool or allocate additional disk space to the sbspaces. For more information, see "Preventing Memory Queues from Overflowing" on page 9-14.

To create row data sbspaces, use the **onspaces -c** utility. 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 path name for an 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 KB). The database server uses this value to calculate the metadata size. The minimum value for AVG_LO_SIZE is 2 KB, which is appropriate for Enterprise Replication in most cases. (The default value of AVG_LO_SIZE is 32 KB.) 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

  Set this parameter to OFF to create an sbspace without logging. Set this parameter to ON to create an sbspace with logging. Use a combination of logging and non-logging sbspaces for Enterprise Replication. For more information, see "Logging Mode for sbspaces."

Set the CDR_QDATA_SBSPACE configuration parameter in the ONCONFIG file to the location of the row data sbspace (**er_sbspace**, in this example). For more information, see "CDR_QDATA_SBSPACE Configuration Parameter" on page B-12.

Logging Mode for sbspaces:

Enterprise Replication uses the default log mode that the sbspace was created with for spooling row data.

Create sbspaces with a default logging mode of ON or OFF according to the types of transactions Enterprise Replication replicates:

- **LOGGING=ON**

  Create sbspaces with LOGGING set to ON to support these situations:

  – Replicated systems with high-availability clusters

    Enterprise Replication must use logging sbspaces for transactions involved in high-availability clusters.

  – Small transactions

    Enterprise Replication uses logging sbspaces for transactions that are less than a page size (2K or 4K) of replicated data.

  For logging sbspaces, performance might be enhanced because logging mode enables asynchronous IO. However, a logging sbspace consumes additional logical-log space.

- **LOGGING=OFF**

  Create sbspaces with LOGGING set to OFF to support replication of large transactions (greater than a page size of replicated data).

  It is recommended that you mirror non-logging sbspaces. For more information, see the chapter on managing disk space in the *IBM Informix Administrator's Guide* and the *IBM Informix Administrator's Reference*.

  For non-logging sbspaces, performance is enhanced on the database server when Enterprise Replication spools to disk because Enterprise Replication writes less data to disk.

Important: Do not change the Enterprise Replication sbspace default log mode while Enterprise Replication is running. To change the default log mode, follow the procedure below.

You can change the default logging mode of the row data sbspace if you have more than one sbspace specified by the CDR_QDATA_SBSPACE configuration parameter.

To change the default logging mode of a row data sbspace:

1. Shut down the database server.
2. Remove the sbspace from the CDR_QDATA_SBSPACE configuration parameter value list.
3. Start the database server in recovery mode.
4. Wait for all the smart large objects to get deleted from the sbspace. Use the **onstat -g smb lod** command to check for smart large objects stored in an sbspace.
5. Change the default logging mode for the sbspace.
6. Add the sbspace name to the CDR_QDATA_SBSPACE configuration parameter value list.
7. Shut down and restart the database server using the **onmode -ky** and **oninit** commands.

Dropping a Spooled Row Data sbspace:

Important: Do not drop an Enterprise Replication row data sbspace using the **onspaces -d -f** (force) command.

You can drop a row data sbspace if you have more than one sbspace specified by the CDR_QDATA_SBSPACE configuration parameter.

To drop a row data sbspace

1. Shutdown the database server.
2. Remove the sbspace from the CDR_QDATA_SBSPACE configuration parameter value list.
3. Start the database server in recovery mode.
4. Wait for all the smart large objects to get deleted from the sbspace. Use the **onstat -g smb lod** command to check for smart large objects stored in a sbspace.
5. Drop the sbspace.

Setting Up the Grouper Paging File

Enterprise Replication uses a grouper paging mechanism for evaluating large transactions. A transaction is large if the portion to be replicated meets at least one of the following conditions:

- It has greater than 5,000 log records.
- It exceeds one fifth the size of the value of the CDR_QUEUEMEM ONCONFIG variable.
- It exceeds one tenth the size of the value of the SHMVIRTSIZE configuration variable.

The location of the sbspace used for the paging file is determined by the first of the following ONCONFIG configuration parameters that is set:

- SBSPACETEMP
- SBSPACENAME
- CDR_QDATA_SBSPACE

The best solution is to set up an unlogged sbspace, as specified by the SBSPACETEMP configuration parameter. All updates to the paging files are unlogged.

The size of the paging sbspace should be at least three times the size of the largest transaction to be processed. This sbspace is also used by the database server for other tasks; consider its overall usage when determining size requirements.

Important: If the paging sbspace fills, Enterprise Replication hangs until you allocate additional disk space to the sbspace. If additional space is unavailable, use the **cdr stop** command to stop 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. You can repair data after replication has failed by using ATS and RIS files. Enterprise Replication examines the specified ATS or RIS file and attempts to reconcile the rows that failed to be applied. See “Repairing Failed Transactions with ATS and RIS Files” on page 8-21 for more information.

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 3-5) and have a conflict resolution rule other than *ignore* or *always-apply* ("Conflict Resolution" on page 3-6) enabled, create the ATS directory on all participating replication systems.

- RIS files

  If you have a conflict resolution role other than *ignore* or *always-apply* enabled, create the RIS directory on all participating replication systems.

The default location for these directories is **/tmp** (UNIX) or **\tmp** (Windows). 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 path names for the ATS and RIS directories cannot be longer than 256 characters.

For information about ATS and RIS, refer to Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1.

Related tasks:

"Enabling ATS and RIS File Generation" on page 9-4

"Customizing the Replication Server Definition" on page 6-6

"Setting Up Failed Transaction Logging" on page 6-11

Preparing the Database Server Environment

To prepare the database server environment, set database server environment variables and configuration parameters, and synchronize the operating system time on all participating database servers.

If you are using high-availability clusters with Enterprise Replication, set up your servers according to the instructions in "Setting Up Database Server Groups for High-Availability Cluster Servers" on page 5-5.

Setting Database Server Environment Variables

Certain environment variables must be set in a replication 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 IBM Informix directory.
- INFORMIXSERVER is set to the name of the default database server.

  For more information, see also "Connect Option" on page A-3.
- INFORMIXSQLHOSTS is set to the full path to the SQLHOSTS file (UNIX) or the SQLHOSTS registry host machine (Windows).
- DELIMIDENT is not set or set to `n`. Enterprise Replication does not allow delimited identifiers.

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

Setting Configuration Parameters

You must set certain configuration parameters before starting Enterprise Replication. You can set some Enterprise Replication configuration parameters dynamically.

In the onconfig file for each database server, make sure that the DBSERVERNAME configuration parameter is set to the correct database server. If you use both the DBSERVERNAME and DBSERVERALIASES configuration parameters, set the DBSERVERNAME configuration parameter to the TCP connection and not to a shared-memory connection. For information about database server aliases, see the *IBM Informix Administrator's Guide*.

In addition, set the following Enterprise Replication configuration parameters in the onconfig file before starting replication:

- CDR_DBSPACE is set to the dbspace for the **syscdr** database. If not set, the root dbspace is used.
- CDR_QUEUEMEM is set to the maximum amount of memory to be used for the send and receive queues.
- CDR_QDATA_SBSPACE is set to the location of the row data sbspace.

  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 is set to generate non-overlapping (unique) values for serial columns across all database servers in the replication environment.

If you want to suppress certain data sync error and warning codes from appearing in ATS and RIS files, you can set the CDR_SUPPRESS_ATSRISWARN configuration parameter.

If you want to customize the number of data sync threads to use, you can set the CDR_APPLY configuration parameter.

If you want to encrypt network communications, make sure that the following configuration parameters are set in the onconfig file for each database server:

- ENCRYPT_CDR is set to 1 or 2 to enable encryption. The default value is 0, which prevents encryption.
- ENCRYPT_CIPHERS specifies which ciphers and cipher modes are used for encryption.
- ENCRYPT_MAC controls the level of Message Authentication Code (MAC) used to ensure message integrity.
- ENCRYPT_MACFILE is set to the full path and file names of the MAC files.
- ENCRYPT_SWITCH is set to the number of minutes between automatic renegotiations of ciphers and keys. (The cipher is the encryption methodology. The secret key is the key used to build the encrypted data using the cipher.)

When replication is active on an instance, you might need to double the amount of lock resources, to accommodate transactions on replicated tables.

By default, if Enterprise Replication detects the potential for a log wrap situation when replication log processing lags behind the current log position, user transactions are blocked. You can configure Enterprise Replication to prevent the blocking of user transactions. Depending on the solutions you need, you might set the following configuration parameters in the onconfig file for each database server:

- CDR_LOG_LAG_ACTION specifies the actions that Enterprise Replication during a potential log wrap situation.
- LOG_STAGING_DIR specifies a directory in which compressed log files are staged.

- CDR_LOG_STAGING_MAXSIZE specifies the maximum size that Enterprise Replication can use to stage log files.
- CDR_MAX_DYNAMIC_LOGS specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.
- DYNAMIC_LOGS specifies that logical logs can be added dynamically.

You can dynamically update the values of Enterprise Replication configuration parameters while replication is active.

Related concepts:

"Row Data sbspaces" on page 4-10

"Serial Data Types and Primary Keys" on page 2-7

"Configuring network encryption for replication servers" on page 4-5

Related tasks:

"Managing Replication Servers" on page 8-1

"Adding a server to the domain by cloning a server" on page 6-5

Related reference:

Appendix B, "Enterprise Replication configuration parameter and environment variable reference," on page B-1

Time Synchronization

Whenever you use replication that requires time stamp, time stamp with a stored procedure, or delete wins conflict resolution, you must synchronize the operating system times of the database servers that participate in the replicate.

All timestamps 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. You should 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.

To synchronize the time on one database server with the time on another database server, use one of the following commands, where *hostname* and *servername* is the name of the remote database server computer.

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. If the operating system times of the database servers do become out of sync or if their times move backward, time stamp or stored procedure conflict resolution might produce failures caused by incorrect time stamps.

Related concepts:

"Conflict Resolution" on page 3-6

"Delete Wins Conflict Resolution Rule" on page 3-12

"Time Stamp Conflict Resolution Rule" on page 3-7

Related tasks:

"Adding a server to the domain by cloning a server" on page 6-5

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 4-26.

Related concepts:

"Update-Anywhere Replication System" on page 3-5

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 that you 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. For information on loading the data, see "Load and unload data" on page 4-24.

Blocking Replication

You might need to block replication so that you can 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 **cdrserver**, **cdrtime**, and **ifx_replcheck** shadow columns 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,ifx_replcheck, col1, ..., colN)
   cdrserver, cdrtime *
   FROM table_name4;
```

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

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

Related concepts:

"Load and unload data" on page 4-24

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 Informix 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 CHAR      stmt[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 Informix ESQL/C programs. Do not use the '$' syntax.

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

Preparing to Replicate User-Defined Types

You must install and register user-defined types on all database servers prior to starting replication.

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 2-16.

Preparing to Replicate User-Defined Routines

You must install and register user-defined routines on all database servers prior to starting replication.

Preparing Tables for Conflict Resolution

To use any conflict-resolution rule other than ignore or always-apply, you must define the shadow columns, **cdrserver** and **cdrtime** in the tables on both the source and target servers involved in replication.

To define the **cdrserver** and **cdrtime** shadow columns when you create a new table, use the WITH CRCOLS clause. For example, the following statement creates a new table named **customer** with a data column named **id** and the two shadow columns:

```
CREATE TABLE customer(id int) WITH CRCOLS;
```

To add the **cdrserver** and **cdrtime** shadow columns to an existing replicated table:

1. Set alter mode on the table by running the **cdr alter --on** command.
2. Alter the table using the ADD CRCOLS clause.
3. Unset alter mode on the table by running the **cdr alter --off** command.

Adding CRCOLS columns to an existing table can result in a slow alter operation if any of the table columns have data types that require a slow alter. If a slow alter operation is necessary, make sure you have disk space at least twice the size of the original table, plus extra log space.

For example, the following statement adds the shadow columns to an existing table named **customer**:

```
ALTER TABLE customer ADD CRCOLS;
```

You cannot drop conflict resolution shadow columns while replication is active. To drop the **cdrserver** and **cdrtime** shadow columns, stop replication and then use the

DROP CRCOLS clause with the ALTER TABLE statement. For example, the following statement drops the two shadow columns from a table named **customer**:

```
ALTER TABLE customer DROP CRCOLS;
```

Related concepts:

"Shadow Columns" on page 2-6

"Shadow Column Disk Space" on page 4-9

Using the WITH CRCOLS Option (SQL Syntax)

"Limited SQL Statements" on page 2-11

Related reference:

Enterprise Replication shadow columns (SQL Syntax)

Preparing Tables for a Consistency Check Index

To improve the speed of consistency checking with an index, you must define the **ifx_replcheck** shadow column in the tables on both the source and target servers involved in replication.

To define the **ifx_replcheck** shadow column when you create a new table, use the WITH REPLCHECK clause. For example, the following statement creates a new table named **customer** with a data column named **id** and the **ifx_replcheck** shadow column:

```
CREATE TABLE customer(id int) WITH REPLCHECK;
```

To add the **ifx_replcheck** shadow column to an existing replicated table:

1. Set alter mode on the table by running the **cdr alter --on** command.
2. Alter the table using the ADD REPLCHECK clause.
3. Unset alter mode on the table by running the **cdr alter --off** command.

Because altering a table to add the **ifx_replcheck** shadow column is a slow alter operation, make sure you have disk space at least twice the size of the original table plus log space.

For example, the following statements add the **ifx_replcheck** shadow column to an existing table named **customer**:

```
ALTER TABLE customer ADD REPLCHECK;
```

To drop the **ifx_replcheck** shadow column, use the DROP REPLCHECK clause with the ALTER TABLE statement. For example, the following statements drop the **ifx_replcheck** shadow column from a table named **customer**:

```
ALTER TABLE customer DROP REPLCHECK;
```

For more information on the CREATE TABLE and ALTER TABLE statements, see the sections in the *IBM Informix Guide to SQL: Syntax*.

Related concepts:

"Shadow Column Disk Space" on page 4-9

"Shadow Columns" on page 2-6

"Limited SQL Statements" on page 2-11

Related tasks:

"Indexing the ifx_replcheck Column" on page 8-19

Related reference:

Enterprise Replication shadow columns (SQL Syntax)

Using the WITH REPLCHECK Keywords (SQL Syntax)

Preparing tables without primary keys

The data columns in your table might not need a primary key. To replicate tables that do not have primary keys, you must add the ERKEY shadow columns. The ERKEY shadow columns are also useful if you plan to update the primary key on your data columns because the ERKEY shadow columns allow you to do so without interrupting replication.

The ERKEY shadow columns are **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**. Enterprise Replication uses a unique index and a unique constraint on these columns in place of a primary key on data columns.

If you create a replicated table through a grid, the ERKEY shadow columns are automatically created and included in the replicate definition.

To enable replication without a primary key:

1. Add the ERKEY shadow columns when you create at table by using the WITH ERKEY keywords with the CREATE TABLE statement. For example, the following statement adds the ERKEY shadow columns to a table named **customer**:

   ```
   CREATE TABLE customer (id int) WITH ERKEY;
   ```

2. Define the replicate. If you define a replicate by using the **cdr define replicate** command, include the **--erkey** option. If you define a template by using the **cdr define template** command, the ERKEY columns are included in the replicate definition automatically.

To add the ERKEY shadow columns to an existing table that you want to start replicating:

1. Run the ALTER TABLE statement with the ADD ERKEY clause. For example, the following statement adds the ERKEY shadow columns to an existing table named **customer**:

```
ALTER TABLE customer ADD ERKEY;
```

Occasionally, you might need to drop the ERKEY shadow columns; for example, if you are reverting to an earlier version of the database server.

To drop the ERKEY shadow columns from a replicated table:

1. Run the **cdr remaster** command without the **--erkey** option.
2. Run the DROP ERKEY clause with the ALTER TABLE statement.

For example, the following statement drops the ERKEY shadow columns from a table named **customer**:

```
ALTER TABLE customer DROP ERKEY;
```

Related concepts:

"Primary Key Constraint" on page 2-7

"Limited SQL Statements" on page 2-11

Related tasks:

"Creating replicated tables through a grid" on page 7-10

"Attaching a New Fragment to a Replicated Table" on page 8-27

Related reference:

"cdr define replicate" on page A-61

"cdr remaster" on page A-119

"cdr change replicate" on page A-32

"cdr define template" on page A-75

Using the WITH ERKEY Keywords (SQL Syntax)

Enterprise Replication shadow columns (SQL Syntax)

Preparing Logging Databases

Databases on all server instances involved in replication must be created with logging. For best results, use unbuffered logging. For more information, see "Unbuffered Logging" on page 2-5.

Related reference:

"cdr start sec2er" on page A-137

Preparing for Role Separation (UNIX)

You can use role separation to allow members of the DBSA group to run Enterprise Replication commands, in addition to the user **informix**.

The DBSA user who runs Enterprise Replication commands must be a member of the DBSA group on all of the replication servers in the domain.

For some Enterprise Replication commands, you must grant the DBSA user additional permissions on tables or files. The following table describes the permissions needed for each command.

Table 4-3. Permissions for the DBSA user

Command	Type of Permission	Tables, Files, or Database
cdr check replicate **cdr check replicateset** **cdr define replicate** **cdr define replicateset** **cdr define template** **cdr realize template** **cdr sync replicate** **cdr sync replicateset**	INSERT UPDATE DELETE	The tables that participate in replication. Must be granted on all replication servers in the domain.

Table 4-3. Permissions for the DBSA user (continued)

Command	Type of Permission	Tables, Files, or Database
The following commands with the **--background** option: • **cdr check replicate** • **cdr check replicateset** • **cdr sync replicate** • **cdr sync replicateset**	CONNECT or INSERT, depending on the object	**sysadmin** database: CONNECT **ph_task** table in the **sysadmin** database: INSERT Must be granted on the database server from which the command is run.
cdr define repair **cdr start repair** **cdr stop repair** **cdr delete repair** The following commands with the **--syncdatasource** option: • **cdr realize template** • **cdr start replicate** • **cdr start replicateset**	INSERT, UPDATE, or DELETE, depending on the table	The following **syscdr** tables: • **rsncjobdef_tab**: INSERT, UPDATE, DELETE • **rsncjobdef**: UPDATE • **rsncprocnames_tab**: INSERT • **rsncjobdeps**: INSERT Must be granted on all replication servers in the domain.
cdr repair **cdr view atsdir** **cdr view risdir**	read	ATS and RIS files Must be granted on the database server on which the files are located.

To update the permissions on a table or database, use the GRANT statement. For example, the following statement grants INSERT and UPDATE permissions on the **rsncjobdef_tab** table to the DBSA member with the user name of **carlo**:

```
GRANT INSERT, UPDATE ON rsncjobdef_tab TO carlo;
```

For more information on the GRANT statement, see the *IBM Informix Guide to SQL: Syntax*.

To update the permissions on ATS and RIS files, use an operating system command, such as the **chown** UNIX command.

Related reference:
"cdr check replicate" on page A-37
"cdr check replicateset" on page A-47
"cdr sync replicate" on page A-161
"cdr sync replicateset" on page A-164
"cdr repair" on page A-122
"cdr view" on page A-169
"cdr realize template" on page A-114
"cdr define replicate" on page A-61
"cdr define replicateset" on page A-69
"cdr start replicate" on page A-131
"cdr start replicateset" on page A-134
"cdr define template" on page A-75

Load and unload data

You can load data into or unload data out of tables in your replication environment in various ways, depending on your circumstances.

If you have not yet set up your replication environment, for loading data, you can use the following tools:

- High-Performance Loader
- onunload and onload Utilities
- dbexport and dbimport utilities
- UNLOAD and LOAD statements
- External tables

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.

Existing replication environment

If you are adding a table to your already existing replication environment, Enterprise Replication provides an initial synchronization feature that allows you to easily bring a new table up-to-date with replication. You can synchronize the new table with data on the source server you specify when you start the new replicate, or when you add a new participant to an existing replicate. You do not need to suspend any servers that are replicating data while you add the new replicate and synchronize it.

If you want to use load and unload tools on tables that are already being replicated, you should block replication while you prepare the table.

If a table that you plan to replicate includes the CRCOLS or REPLCHECK shadow columns, the statements that you use for unloading the data must explicitly name the shadow columns. If you use the SELECT statement with * FROM *table_name* to the data to unload, the data from the shadow columns is not unloaded. To include the shadow columns in the unloaded data, explicitly name them. For example, use a statement like the following:

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

If a table that you plan to replicate includes ERKEY shadow columns, you cannot unload and then load the data from these columns and preserve the original values. If you need to preserve the values of the ERKEY shadow columns, use synchronization to propagate the values.

Related concepts:

"Blocking Replication" on page 4-17

"Setting Up Database Server Groups for High-Availability Cluster Servers" on page 5-5

"Shadow Columns" on page 2-6

Related tasks:

Moving data with external tables (Administrator's Guide)

"Initially Synchronizing Data Among Database Servers" on page 6-20

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 contains shadow columns, you must:

- Include all the shadow column names in your map when you load the data.
- Use express mode to load data that contains shadow 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 *IBM Informix High-Performance Loader User's Guide*.

onunload and onload Utilities

You can use the **onunload** and **onload** utilities to unload and load an entire table.

If you want to unload selected columns of a table, you must use either the UNLOAD statement or the HPL.

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

If you use the **onload** utility while replication is active, you must synchronize the data after you finish loading the data.

Related concepts:

The onunload and onload utilities (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.

Related concepts:

The dbexport and dbimport utilities (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 all shadow columns in your map when you unload the data.
- List the columns that you want to load in the INSERT statement and explicitly include the shadow columns in the list when you load your data.

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

Data Preparation Example

The following examples show how to add a new participant (**delta**) to an existing replicate by two different methods:

- Using the **cdr start replicate** command

  This method is simple and can be done while replication is online.
- Using the LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION statements.

  If you use HPL, this method can be faster for a large table.

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.

Using the cdr start replicate Command

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

   ```
   cdr sta rep zebra g_delta -S g_alpha -e delete
   ```

   The **-S g_alpha** option specifies that the server **alpha** be used as the source for data synchronization.

The **-e delete** option indicates that if there are rows on the target server, **delta**, that are not present on the synchronization data server (**alpha**) then those rows are deleted

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

Using LOAD, UNLOAD, and BEGIN WORK WITHOUT REPLICATION

When you add a new participant to an existing replicate, you can unload and load data without replication.

To add a new participant to an existing replicate

1. Add the server **delta** to the Enterprise Replication domain. 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 "P dbname@g_delta:owner.table1" \
   "select * from 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 **delta** on **alpha**, **beta**, and **gamma**.

   ```
   cdr sus ser g_delta 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 queuing 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.

   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 transactions. 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**.

   ```
   cdr res ser g_delta g_alpha g_beta g_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. Transactions can abort because **alpha**, **beta**, and **gamma** started queuing data from **delta** in step 4. However, those same transactions might have been moved in steps 5 and 7.

You must declare replication on server **delta** and then immediately suspend replication 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.

Chapter 5. Using High-Availability Clusters with Enterprise Replication

In This Chapter

This chapter covers how to include other data replication solutions, such as high-availability data replication, in your Enterprise Replication system. The following topics are covered:

- The design of a high-availability cluster replication system
- Preparing a high-availability cluster database server
- Managing Enterprise Replication with a high-availability cluster

For a complete description of data replication, see the *IBM Informix Administrator's Guide*.

High-Availability Replication System

You can combine IBM Informix Enterprise Replication and high-availability solutions to create a high-availability replication system in which a critical read-write database server in an IBM Informix Enterprise Replication system maintains a backup server with a high-availability cluster primary server. A high-availability cluster consists of a primary server and:

- a single HDR secondary server
- zero or more SD (shared disk) secondary servers
- zero or more RS (remote standalone) secondary servers

HDR consists of a primary server and a single HDR secondary server that are tightly coupled; transactions on the primary are not committed until the log records containing the transactions are sent to the HDR secondary server. SD secondary servers do not maintain a copy of the physical database on its own disk space; rather, they share disks with the primary server. SD secondary servers can be quickly and easily promoted to be the primary server if the primary goes offline. The third type of secondary server, remote standalone (RS) secondary server, can also be used in a high-availability solution that includes IBM Informix Enterprise Replication. For a description of the differences between HDR secondary servers, SD secondary servers, and RS secondary servers, see the *IBM Informix Administrator's Guide*.

A high-availability cluster consists of two types of database servers: the primary database server, which receives updates, and one or more secondary copies of the primary database server. A secondary server is a mirror image of the primary server and is in perpetual recovery mode, applying logical-log records from the primary server.

The secondary server does not participate in IBM Informix Enterprise Replication; it receives updates from the primary server. If the primary server in a high-availability cluster becomes unavailable, one of the secondary servers takes over the role of the primary server. Using Connection Manager, you can specify which secondary server should take over in the event of a failure of the primary server.

High-availability replication systems are most useful for replication systems in which the failure of a critical server prevents other servers from participating in replication. The examples in this chapter show how to use HDR with IBM Informix Enterprise Replication, but SD or RS secondary servers could be used equally effectively instead. Figure 5-1 illustrates the combination of a primary-target IBM Informix Enterprise Replication system with a high-availability cluster configured with an HDR secondary server.

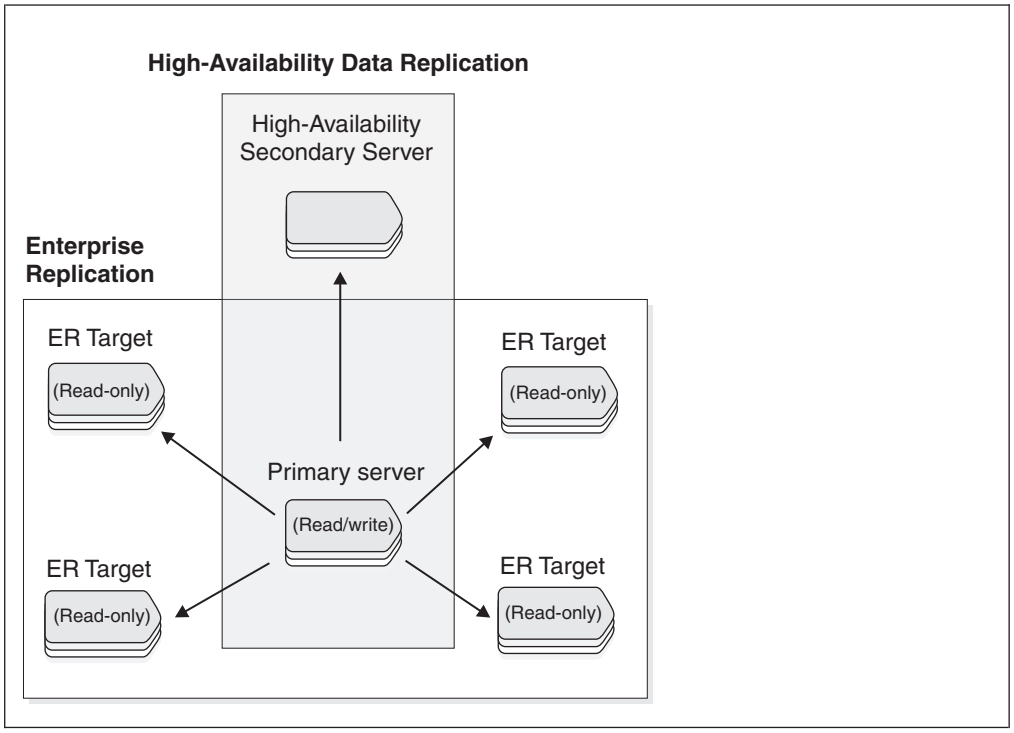


Figure 5-1. Using a High-Availability Cluster with a Primary-Target Replication System

If the primary server fails, the secondary server is set to standard mode, the target database connections are redirected to it, and IBM Informix Enterprise Replication continues, as illustrated in Figure 5-2 on page 5-3.

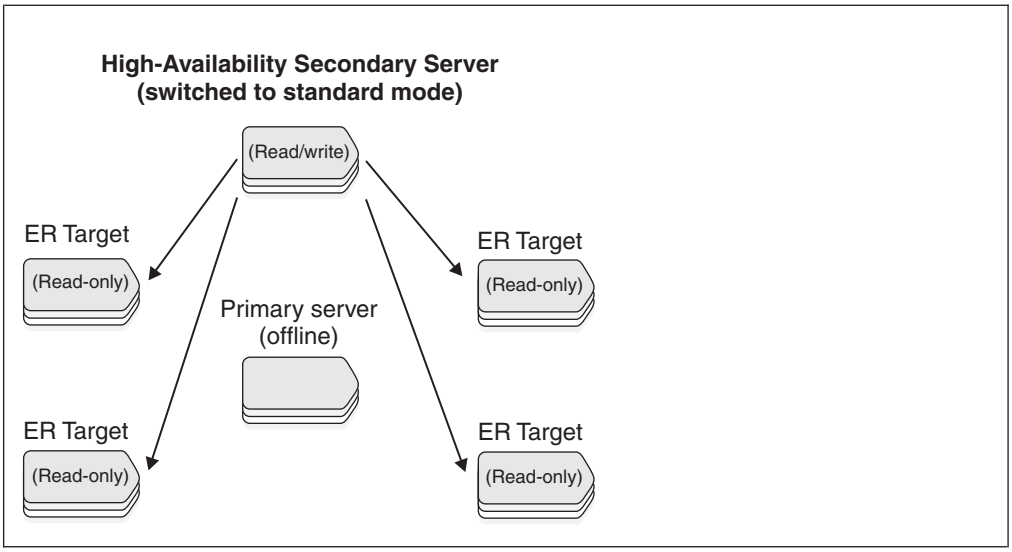


Figure 5-2. Redirection to the Secondary Database Server

In an update-anywhere replication system, you can use HDR with any server for which you need high availability, as illustrated in Figure 5-3.

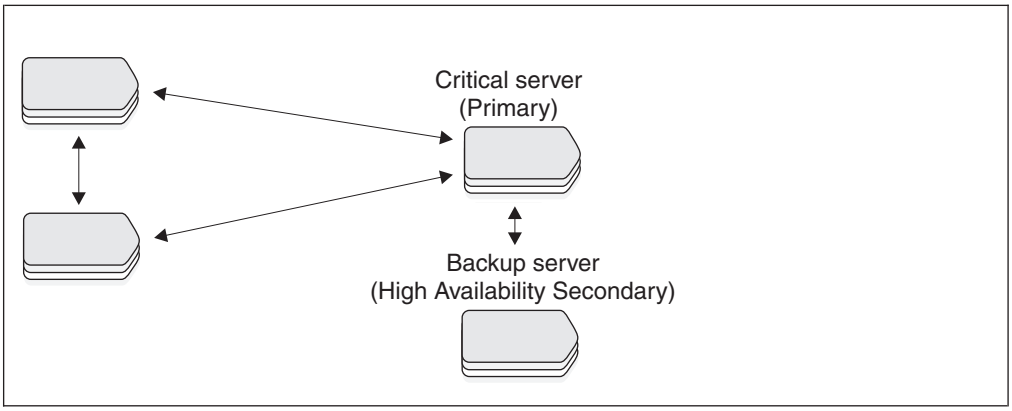


Figure 5-3. Using High Availability with an Update-Anywhere Replication System

Using high-availability clusters with IBM Informix Enterprise Replication is particularly effective when you use a hierarchical or a forest of trees topology.

Related concepts:

"Update-Anywhere Replication System" on page 3-5

High-Availability Clusters in a Hierarchical Tree Topology

With a hierarchical tree topology, parent servers are good candidates for using high-availability clusters to provide backup servers.

The following example is based on the example in Figure 3-7 on page 3-17.

If **China** fails, then **Beijing** and **Shanghai** can no longer replicate with other servers in the replication system; **Guangzhou** and **Chengdu** can replicate only with each other. However, if **China** was part of a high-availability cluster, when it failed, the secondary server would replace it and replication would continue, as illustrated in Figure 5-4 on page 5-4.

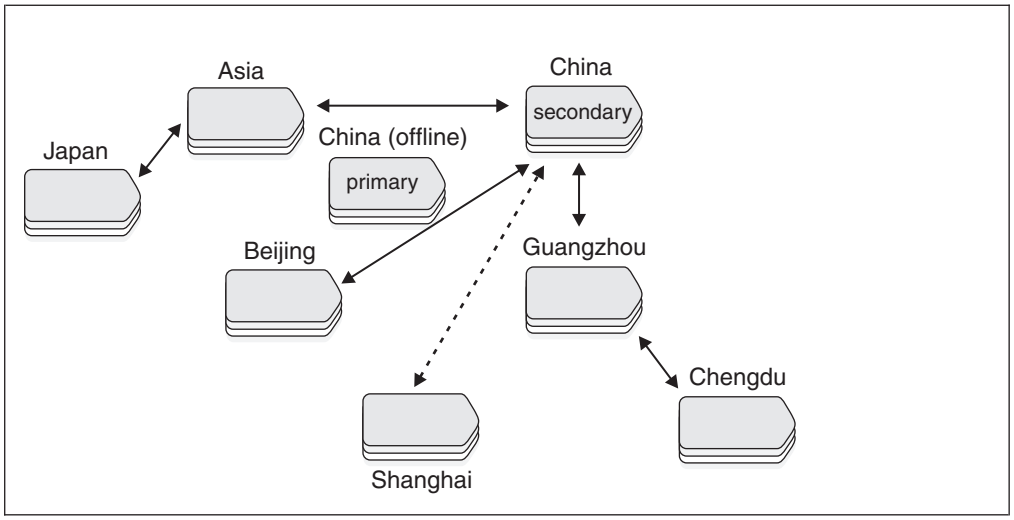


Figure 5-4. Hierarchical Tree Topology with HDR

In this example, **Asia** and **Guangzhou**, which are also parent servers, might also benefit from using a high-availability cluster to ensure high availability.

Using high-availability clusters in a forest of trees topology

Use a high-availability cluster to ensure that all servers retain access to the replication system in a forest of trees topology.

For example, in Figure 3-8 on page 3-18, **Asia**, **Europe**, **China**, and **Guangzhou** should use high-availability clusters to provide backup servers, as illustrated in Figure 5-5 on page 5-5.

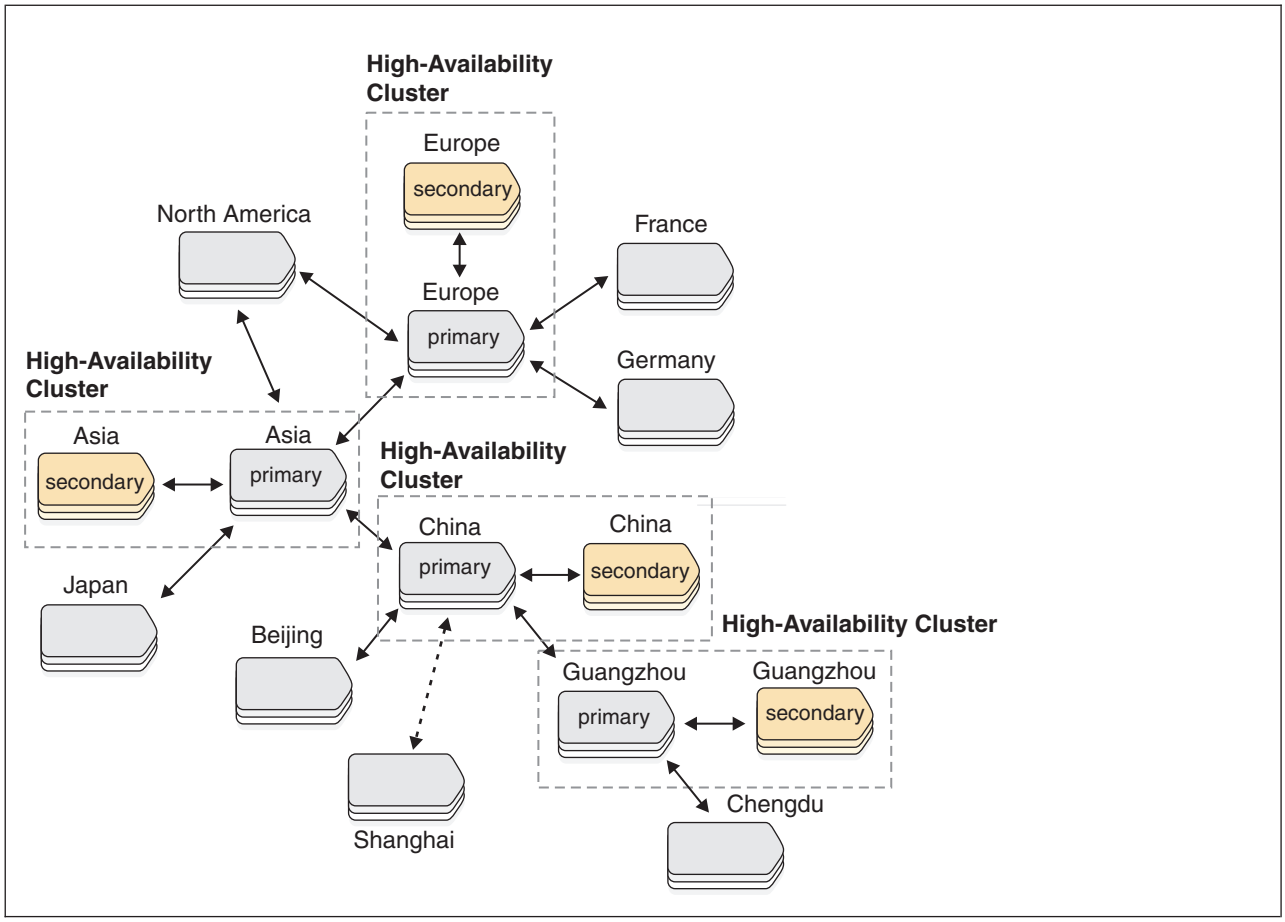


Figure 5-5. High-Availability Clusters in a Forest-of-Trees Topology

Setting Up Database Server Groups for High-Availability Cluster Servers

When defining a high-availability cluster within Enterprise Replication, the cluster must appear to be a single logical entity within the replication domain. Define the servers within the same database server group in the `sqlhosts` file.

For example, Figure 5-6 on page 5-6 illustrates two Enterprise Replication nodes, one of which is an HDR pair.

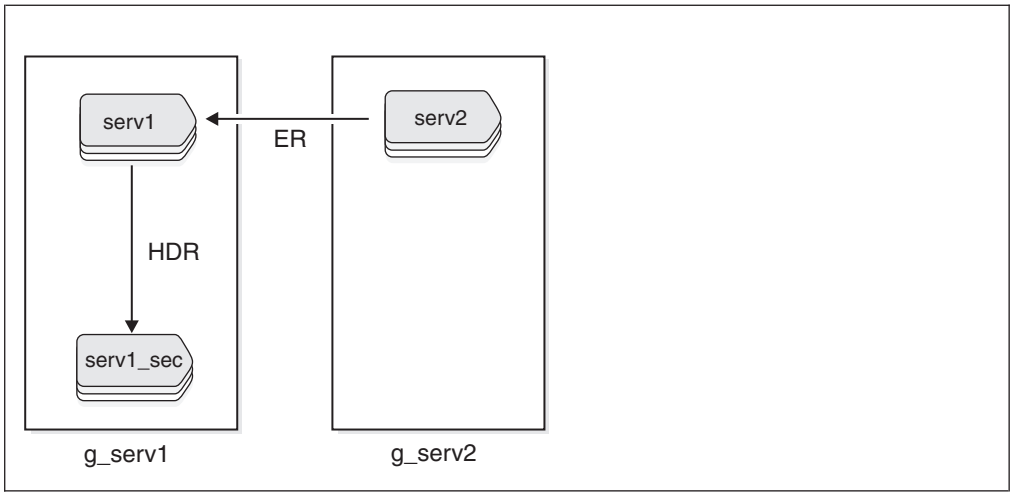


Figure 5-6. Database Server Groups for Enterprise Replication with HDR

In this example, the HDR pair consists of the primary server, **serv1**, and the secondary server, **serv1_sec**. These two servers belong to the same database server group, **g_serv1**. The non-HDR server, **serv2**, belongs to the database server group **g_serv2**. The following example displays the `sqlhosts` file for this configuration:

```
#dbservername    nettype    hostname     servicename  options
 g_serv1         group      -            -            i=1
 serv1           ontlitcp   machine1pri  port1        g=g_serv1
 serv1_sec       ontlitcp   machine1sec  port1        g=g_serv1
 g_serv2         group      -            -            i=2
 serv2           ontlitcp   machine2     port1        g=g_serv2
```

Important: If you use the g=server option in the group member definition, you can put the definition anywhere in the `sqlhosts` file.

Either HDR or Enterprise Replication can be set up first on the HDR pair **serv1** and **serv1_sec**, but Enterprise Replication **cdr** commands must be run only on the primary server. If any **cdr** commands are attempted on the secondary server, a –117 error is returned: `Attempting to process a cdr command on an HDR secondary server`.

Related concepts:

"Load and unload data" on page 4-24

"Database Server Groups" on page 4-2

Managing Enterprise Replication with High-Availability Clusters

This section describes how to manage Enterprise Replication with HDR in the following areas:

- Failure of the primary server in a high-availability cluster
- Performance considerations

Related concepts:

"Database Server Groups" on page 4-2

Failure of the Primary Server in a High-Availability Cluster

If the primary server within a high-availability cluster fails, and you have configured Connection Manager failover arbitration, one of the secondary servers

you have designated will take over the role of the primary server. Enterprise Replication will connect to the new primary server and no action is required.

If Connection Manager failover arbitration is not configured, a secondary server can be converted to the primary server using the **onmode -d make primary** command. Enterprise Replication will connect to the new primary server. If the primary server within a high-availability cluster fails, the secondary server can be switched to standard mode by executing the **onmode –d standard** command. However, you must manually start Enterprise Replication by executing the **cdr start** command on that server. This is necessary to prevent Enterprise Replication from starting on all servers in cluster. Table 5-1 shows how to switch the secondary server to standard mode.

Table 5-1. Switching the Secondary Server to Standard Mode

Step	On the Primary	On the Secondary
1.	The server becomes unavailable.	
2.		**onmode command** onmode -d standard
3.		**cdr command** cdr start

If you need to start the primary server while Enterprise Replication is running on the secondary server, use the **oninit –D** command to prevent Enterprise Replication and HDR from starting on the primary server.

If the problem has been resolved on the primary server and you wish to reestablish it as the primary server, then first stop Enterprise Replication on the secondary server. Otherwise, Enterprise Replication attempts to restart on the primary server while it is still active on the secondary server. Table 5-2 shows how to reestablish the primary server.

Table 5-2. Reestablishing the Primary Server

Step	On the Primary	On the Secondary
1.		**cdr command** cdr stop
2.		**onmode command** onmode -s
3.		**onmode command** onmode -d secondary
4.	oninit	
5.	**cdr command** cdr start	

If you want to split an active cluster into two stand-alone servers, then you must be careful to avoid Enterprise Replication starting on either server after they are split. To prevent Enterprise Replication and high availability from running, start the database servers with the **oninit –D** command.

If you remove a server from a cluster, use the **cdr delete server –force** command to eliminate Enterprise Replication from that server. For example, the two HDR

servers are being split and the secondary server is to be used for reporting purposes. After the report processing is complete, HDR can be reestablished. “cdr delete server” on page A-81 shows how to remove a secondary server from a high-availability cluster and Enterprise Replication.

Table 5-3. Removing the Secondary Server from a cluster and ER

Step	On the Primary	On the Secondary
1.	**onmode command** onmode -d standard	**onmode command** onmode -d standard
2.		**cdr command** cdr delete server -f *server_name*

If the HDR primary server has problems communicating to its secondary server, Enterprise Replication is in a suspended state until one of the following actions is taken:

- Resolve the connection problem between HDR pairs.
- Convert the primary server to standard mode.

For more information on managing high-availability clusters, see the *IBM Informix Administrator's Guide*.

Related reference:

“cdr delete server” on page A-81

Connection Manager with Enterprise Replication and clusters

You must use different Connection Managers to manage connections for Enterprise Replication replicate sets and for clusters.

When you configure a Connection Manager, you must specify whether to manage a replicate set or a cluster by setting the TYPE keyword to either REPLSET or CLUSTER. A Connection Manager for a replicate set cannot route client connections to a secondary server. Similarly, a Connection Manager for a cluster cannot route client connections to a different replication server.

The following illustration shows two replication servers that are also primary servers in clusters. One cluster has an HDR secondary server and the other cluster has two shared disk secondary servers. Three instances of Connection Manager manage client connections for the clusters and the replication domain.

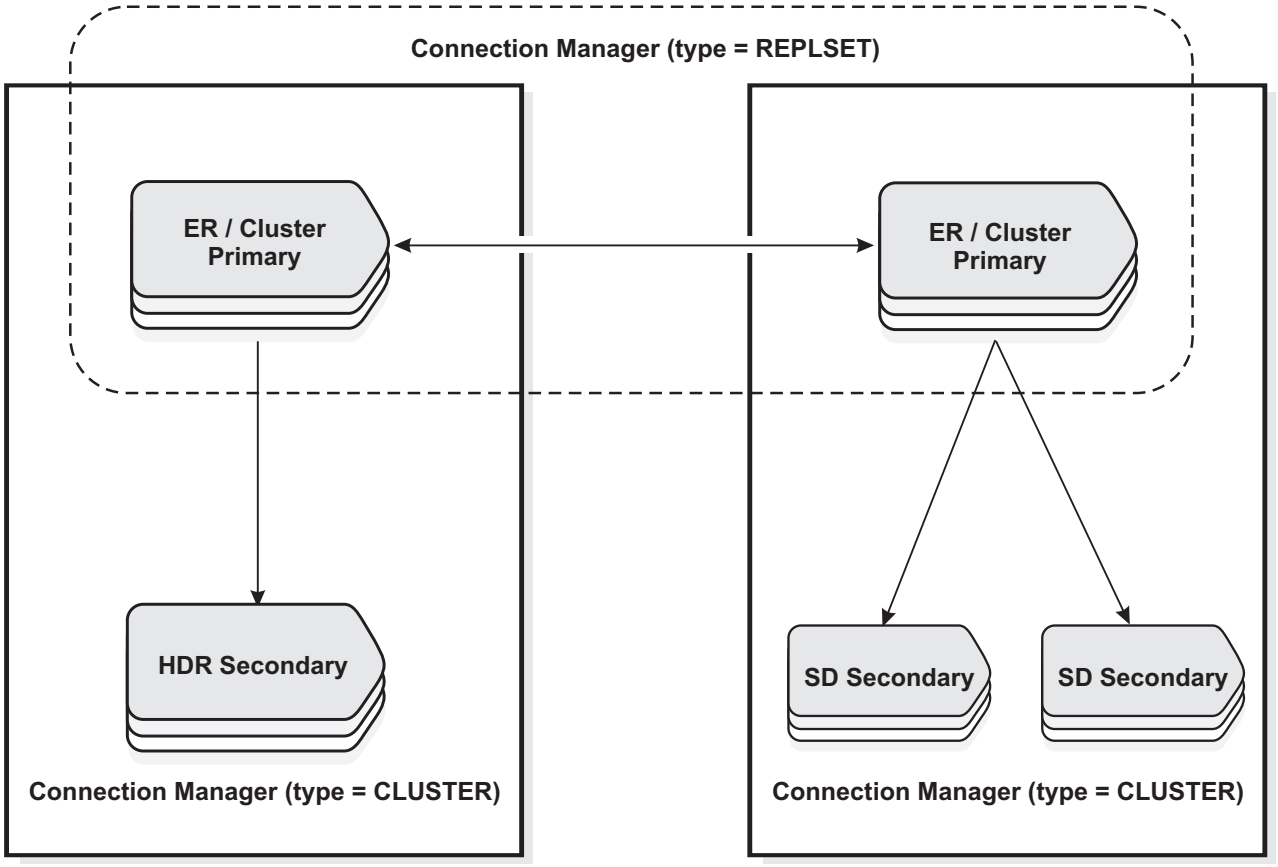


Figure 5-7. Connection Managers needed when replication servers are also primary servers in a cluster

Performance Considerations

When Enterprise Replication is running on an HDR pair, some operations cannot be performed until the logs are shipped to the secondary server. This delay prevents possible inconsistency within the Enterprise Replication domain during an HDR switch-over to a secondary server. Consequently, there is a slight increase in replication latency when Enterprise Replication is used with HDR. You can control this latency increase by setting the DRINTERVAL configuration parameter to a low value.

Alternatively, using SD secondary servers instead of HDR can decrease replication latency.

Chapter 6. Defining Replication Servers, Replicates, Participants, and Replicate Sets

These topics describe the steps defining and starting Enterprise Replication.

To define and start replication:

1. Initialize the database server.
2. Create a replication domain by defining replication servers.
3. Configure replication by defining replicates, and optionally grouping replicates into a replicate set. The *replicate* definition includes information about the participants, replication options, frequency, and conflict-resolution rules and scope.
4. Specify the data to replicate by defining participants. A *participant* definition specifies the data (database, table, and columns) that should be replicated.
5. Synchronize the data among the replicates.

Starting Database Servers

The database server must be online before you can define it as a replication server.

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

Operating System	Command
UNIX	**oninit**
Windows	**start** *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 server operating modes in the *IBM Informix Administrator's Guide*.

Defining Replication Servers

You must define a replication server to create a replication domain or to add a server to an existing domain.

The database server must be online.

The CDR_QDATA_SBSPACE configuration parameter must be set to a valid value.

You must be the Enterprise Replication server administrator to define the replication server.

You can define replication servers using two different methods:

- The **cdr** utility
- Cloning

To define the replication server in a new domain by using the **cdr** utility, use the **cdr define server** command to connect to the database server and specify the database server group name. For example, the following command connects to a server called **stan** and creates a domain containing the database server group **g_stan**:

```
cdr define server  --connect=stan --init g_stan
```

The **--init** option specifies the database server group to add to the replication domain. If the INFORMIXSERVER environment variable is not set to the server that you are defining, specify the **--connect=***server_name* option. You can also configure replication attributes for the server.

To define a replication server in an existing domain by using the **cdr** utility, include the **--sync=***sync_server* option with the **cdr define server** command to synchronize the global catalog with an existing server. For example, the following command adds a server group named **g_oliver** to the domain created in the previous command, using **g_stan** as the synchronization server:

```
cdr define server  --connect=oliver --init g_oliver --sync=g_stan
```

You can specify any existing server in the domain, however, if you define a server as a nonroot or a leaf server, then the synchronization server becomes the parent of the new server. For example, if you add a server **kauai** as a leaf server and want its parent to be **hawaii**, then specify **hawaii** with the **--sync** option.

Related concepts:

"Row Data sbspaces" on page 4-10

"Choosing a Replication Network Topology" on page 3-15

"Enterprise Replication Server Administrator" on page 2-1

Related tasks:

"Setting Up Failed Transaction Logging" on page 6-11

Related reference:

"cdr define server" on page A-71

"cdr define replicate" on page A-61

Creating a new domain by cloning a server

You can create a new replication domain by cloning a server and then converting the two Informix database servers to replication servers. Use cloning and conversion if you want to set up replication based on the data on a source server that is not yet running Enterprise Replication.

Because the source server does not have Enterprise Replication defined, you use the **ifxclone** utility to create a cluster containing a primary server and remote stand-alone (RS) secondary server. The conversion process converts the cluster to a pair of replication servers in a new domain.

To create a new domain with two replication servers:

1. On the source server, prepare the server environment for Enterprise Replication, such as configuring `sqlhosts` information and setting the necessary configuration parameters.
2. On both servers, complete the **ifxclone** prerequisites for all servers, such as setting the required configuration parameters and environment variables.
3. On the target server, complete the **ifxclone** prerequisites for an RS secondary server, such as creating all of the chunks that exist on the source server.

4. On the target server, run the **ifxclone** command with the **--disposition=RSS** option to clone the data and the configuration of the source server onto the target server. Do not include the **--useLocal** option.
5. On the source server, run the **cdr check sec2er** command to determine if conversion to replication servers is possible.
6. Solve any error conditions identified by the **cdr check sec2er** command and rerun it until its output indicates that conversion will be successful. You can also solve warning conditions.
7. On the source server, run the **cdr start sec2er** command to convert both servers to replication servers and create a new replication domain.

To add other servers to the domain, you can clone a replication server.

Related concepts:

Chapter 4, "Preparing the Replication Environment," on page 4-1

Related tasks:

"Adding a server to the domain by cloning a server" on page 6-5

Related reference:

The ifxclone utility (Administrator's Reference)

Example of creating a new replication domain by cloning

This is an example of creating a new replication domain based on the data and configuration on a source database server that does not have replication defined. The three additional replication servers in the domain are added by cloning the source server.

This example creates a replication domain and grid that contain four replication servers: **serv1**, **serv2**, **serv3**, **serv4**. Each server computer has the Informix database server installed. The source server contains the **stores_demo** database.

1. On the **serv1** server, set the CDR_QDATA_SBSPACE configuration parameter.
2. On the **serv1** server, set the value of the ENABLE_SNAPSHOT_CLONE configuration parameter to 1 in the `onconfig` file.
3. On the **serv1** server, add the following `sqlhosts` information about **serv1** and **serv2**:

```
gserv1     group    -                    -     i=143
serv1      ontlitcp ny.usa.com           1230  g=gserv1
gserv2     group    -                    -     i=144
serv2      ontlitcp tokyo.japan.com      1231  g=gserv2
```

4. On both the **serv1** and **serv2** servers, complete the **ifxclone** prerequisites for all servers, such as setting the required configuration parameters and environment variables.

   Set these environment variables:

   - **INFORMIXDIR**
   - **INFORMIXSERVER**
   - **INFORMIXSQLHOSTS**
   - **ONCONFIG**

   Set these configuration parameters to the same values on both servers:

   - DRAUTO
   - DRINTERVAL
   - DRTIMEOUT
   - LOGBUFF

- LOGFILES
- LOGSIZE
- LTAPEBLK
- LTAPESIZE
- ROOTNAME
- ROOTSIZE
- PHYSBUFF
- PHYSFILE
- STACKSIZE
- TAPEBLK
- TAPESIZE

5. On the **serv2** server, create all of the chunks that exist on the **serv1** server.
6. On the **serv2** server, run the **ifxclone** command with the **--disposition=RSS** option to clone the data and the configuration of the **serv1** server onto the **serv2** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv2 --targetIP=192.168.0.2
--targetPort=1231 --disposition=RSS
```

7. On the **serv1** server, run the **cdr check sec2er** command to determine if conversion to replication servers is possible:

```
$cdr check sec2er -c gserv1 gserv2
Secondary conversion to ER is possible.
```

8. On the **serv1** server, run the **cdr start sec2er** command to convert both servers to replication servers, create a new replication domain, create and start replicates based on all the tables on the **serv1** server:

```
cdr start sec2er -c gserv1 gserv2
```

9. On the **serv3** and **serv4** servers, provision chunk paths and other storage to the same paths and at least the same sizes as on the **serv1** server.
10. On the **serv3** server, run the **ifxclone** command with the **--disposition=ER** option to clone the data and the configuration of the **serv1** server onto the **serv3** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv3 --targetIP=192.168.0.3
--targetPort=1232 --disposition=ER
```

11. On the **serv4** server, run the **ifxclone** command with the **--disposition=ER** option to clone the data and the configuration of the **serv1** server onto the **serv4** server:

```
ifxclone --trusted --source=serv1 --sourceIP=192.168.0.1
--sourcePort=1230 --target=serv4 --targetIP=192.168.0.4
--targetPort=1233 --disposition=ER
```

12. Edit the `sqlhosts` files on all four servers so that they each have the following information:

```
gserv1     group    -                     -     i=143
serv1      ontlitcp ny.usa.com            1230  g=gserv1
gserv2     group    -                     -     i=144
serv2      ontlitcp tokyo.japan.com       1231  g=gserv2
gserv3     group    -                     -     i=145
serv3      ontlitcp rome.italy.com        1232  g=gserv3
gserv4     group    -                     -     i=146
serv4      ontlitcp perth.australia.com   1233  g=gserv4
```

Related reference:

"cdr start sec2er" on page A-137

"cdr check sec2er" on page A-54

The ifxclone utility (Administrator's Reference)

Adding a server to the domain by cloning a server

You can add a replication server to an existing replication domain by using the **ifxclone** utility to clone an existing replication server onto a target database server.

Enterprise Replication must be active on the source server. The source server should not have any stopped or suspended replicates or any shadow replicates defined.

You must be user **informix** or member of the **informix** group to run the **ifxclone** utility.

IBM Informix database software must be installed on the target server.

Cloning a server defines replication on the target server, copies the data, and adds the target server to all replicates in which the source server participates. The `onconfig` file and the **sqlhosts** file are copied from the source server to the target server and updated with the target server information.

To clone a replication server by using the **ifxclone** utility:

1. On the source server, set the value of the ENABLE_SNAPSHOT_COPY configuration parameter to 1 in the `onconfig` file.
2. On the target server, create the following directories, if they exist on the source server. The directories must be the same on both servers:
   - ATS and RIS directories
   - Log staging directory
3. On the target server, synchronize the system clock with the source server.
4. On the target server, provision chunk paths and other storage to the same paths and at least the same sizes as on the source server. Ensure that the target server has at least as much memory and disk space resources as the source server.
5. On the target server, run the **ifxclone** command. You must provide the following information to the **ifxclone** utility:
   - Source server name
   - Source IP address
   - Source port
   - Target server name
   - Target IP address
   - Target port

   Include the **--disposition=ER** option.

   If the source server replicates serial columns, use the **--configParam** option to set the value of the CDR_SERIAL configuration parameter to ensure that serial values do not conflict between replication servers. The **ifxclone** utility has the following format for cloning a replication server:

```
ifxclone --source=source_name --sourceIP=source_IP
--sourcePort=source_port --target=target_name
 --targetIP=target_IP --targetPort=target_port
--disposition=ER
```

6. On all other replication servers in the domain, edit the **sqlhosts** file to add entries for the new replication server.

Related concepts:

"Time Synchronization" on page 4-16

Related tasks:

"Setting Configuration Parameters" on page 4-14

"Creating a new domain by cloning a server" on page 6-2

"Adding a server to a grid by cloning" on page 7-5

Related reference:

The ifxclone utility (Administrator's Reference)

Customizing the Replication Server Definition

You can specify replication attributes of a server when you define it.

When you define a replication server, you can specify the following attributes in the **cdr define server** command:

- 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 the **--idle=***timeout* option.

  You can later change the values of this attribute with the **cdr modify server** command.
- Specify the location of the ATS and RIS directories.

  To use ATS, specify the directory for the Aborted Transaction Spooling (ATS) files for the server using **--ats=***dir* or**--ris=***dir* . To prevent either ATS or RIS file generation, set the directory to **/dev/null** (UNIX) or **NUL** (Windows).

  You can later change the values of these attributes with the **cdr modify server** command.
- Specify the format of the ATS and RIS files.

  Use the **–atsrisformat=***type* option to specify whether the ATS and RIS files are generated in text format, XML format, or both formats.

  You can later change the values of this attribute with the **cdr modify server** command.
- Specify the type of server if you are using hierarchical replication:
  - To specify the server as a nonroot server, use the **--nonroot** option.
  - To specify the server as a leaf server, use the **--leaf** option.

  If neither **--leaf** nor **--nonroot** is specified, the server is defined as a root server. The parent server is the server specified by the **--sync=***sync_server* option.

Related concepts:

"Choosing a Replication Network Topology" on page 3-15

Related tasks:

"Creating ATS and RIS Directories" on page 4-13

Related reference:

"cdr define server" on page A-71

"Global Catalog" on page 2-3

"cdr modify server" on page A-112

Defining Replicates

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

You can provide the following information in the replicate definition:

- Participants
- Create as a master replicate
- Conflict resolution rules and scope
- Replication frequency
- Error logging
- Replicate full rows or only changed columns
- IEEE or canonical message formats
- Database triggers
- Code set conversion between replicates

After you define the replicate and participants, you must manually start the replicate using the **cdr start replicate** command.

Participant definitions

You must define a participant for each server involved in the replicate definition using the **cdr define replicate** command. Each participant in a replicate must specify a different database server.

Each participant definition includes the following information:

- Database server group name
- Database in which the table to be replicated resides
- Table name
- Table owner
- SELECT statement and optional WHERE clause

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

  To include the ERKEY shadow columns in the participant definition, use the **--erkey** option with the **cdr define replicate** command.

Restriction: 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 for when x is between 50 and 100 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.

Related concepts:

"Primary-Target Replication System" on page 3-1

Related reference:

"Participant and participant modifier" on page A-4

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.

Defining Master Replicates

To guarantee consistency between the nodes in your Enterprise Replication environment, you can define *master replicates* using the **cdr define replicate** command with the **--master** option. Dictionary information is then stored about replicated column attributes for the participant you specify. This enables Enterprise Replication to check for consistency between this master definition and local participant definitions. Checks are performed when the replicate is defined and each time a new participant is added to the replicate, thus avoiding runtime errors. Verification also occurs each time the master replicate is started on a server.

Defining a replicate as a master replicate provides several advantages:

- Ensures data integrity by verifying that all participants in the replicate have table and replicated column attributes that match the master replicate definition.
- Provides automatic table generation on participants that do not already contain the table specified in the master replicate. However, Enterprise Replication cannot create tables with user-defined data types.
- Allows alter operations on the replicated tables. For more information, see "Alter, Rename, or Truncate Operations during Replication" on page 8-22.

When you define a master replicate, you must specify a participant that is on the server for which you are running the command. This participant is used to create the dictionary information for the master replicate. If you specify additional participants in the **cdr define replicate** command, they are verified against the master definition and added to the replicate if they pass validation. If any participant fails validation, the **cdr define replicate** command fails and that participant is disabled.

The master replicate options, described in subsections below, are:

- **--name**

  Use this option to create a strict master replicate that supports alter operations and remastering.
- **--empty**

  Use this option to create an empty master replicate and add participants later.

Related reference:

"cdr define template" on page A-75

"cdr define replicate" on page A-61

Master Replicate Verification

Enterprise Replication verifies the following information about a participant when the participant is added to the master replicate:

- The participant contains all replicated columns.
- The replicated columns in the participant have the correct data types. For columns that are user-defined data types, only the names of the data types are verified.
- Optionally, the replicated columns in the participant have the same column names as the master replicate.

Creating Strict Master Replicates

You can create a strict master replicate in which all participants have the same replicated column names by using the **--name=y** option. This option specifies that when the master replicate verification is done for a new participant, that the column names on the participant must be identical to the column names of the master replicate. Strict master replicates allow you to perform the following tasks:

- Alter operations on replicated tables. For more information, see "Alter, Rename, or Truncate Operations during Replication" on page 8-22.
- Remastering by using the **cdr remaster** command. For more information, see "Remastering a Replicate" on page 8-27.

You can modify an existing master replicate to remove name verification by using the **--name=n** option of the **cdr modify replicate** command.

Related reference:

"cdr modify replicate" on page A-108

Creating Empty Master Replicates

You can create an empty master replicate by using the **--empty** option. This option allows you to specify a participant as the basis of the master replicate but not include that participant in the replicate. Creating an empty replicate can be convenient in large environments in which you later add all participants using scripts.

When you define an empty master replicate, you must specify only one participant in the **cdr define replicate** command. This participant is used to create the master dictionary information but is not added to the replicate.

The **--empty** option is only supported for master replicates, you cannot use it without the **--master** option.

Defining Shadow Replicates

A *shadow replicate* is a copy of an existing, or primary, replicate. Enterprise Replication uses shadow replicates to manage alter and repair operations on replicated tables. You must create a shadow replicate to perform a manual remastering of a replicate that was defined with the **-n** option. See "Resynchronizing Data Manually" on page 8-22 for information about how you

can repair, or remaster, your replicated data. After creating the shadow replicate, the next step in manual remastering is to switch the primary replicate and the shadow replicate using the **cdr swap shadow** command.

You create a shadow replicate using the **cdr define replicate** command with the **--mirrors** option, as described in "cdr define replicate" on page A-61.

When you define a shadow replicate, its state is always set to the same state as the primary replicate. If you change the state of the primary replicate, all its shadow replicates' states are also changed to the same state.

You cannot delete a primary replicate if it has any shadow replicates defined. You must first delete the shadow replicates, and then the primary replicate.

You cannot modify a primary replicate (using the **cdr modify replicate** command) if it has any shadow replicates defined. Also, you cannot modify shadow replicates directly.

You cannot suspend or resume a primary replicate (using the **cdr suspend replicate** or **cdr resume replicate** command) if it has any shadow replicates defined. Also, you cannot suspend or resume shadow replicates directly. If the primary replicate and its shadow replicates are part of an exclusive replicate set, you can suspend or resume the entire replicate set using the **cdr suspend replicate** or **cdr resume replicate** command.

You cannot add a participant to a shadow replicate:

- If the participant is not part of the primary replicate's definition
- After remastering the replicate

If the primary replicate is part of an exclusive replicate set, any shadow replicates you define are automatically added to that replicate set.

If you add a primary replicate to an exclusive replicate set, all its shadow replicates are also automatically added. If you delete a primary replicate from an exclusive replicate set, all its shadow replicates are also automatically deleted.

Specifying Conflict Resolution Rules and Scope

You specify the conflict resolution rule in the replicate definition.

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

- **always**
- **deletewins**
- **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.

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

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

Related concepts:

"Update-Anywhere Replication System" on page 3-5

"Conflict Resolution Rule" on page 3-6

"Conflict Resolution Scope" on page 3-14

Related reference:

"cdr define replicate" on page A-61

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-26.

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 6-18.

Setting Up Failed Transaction 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.

To configure your replicate to use ATS and RIS

1. Set up the ATS and RIS directories.
2. Specify the location of the ATS and RIS directories when you define your server.
3. Specify that the replicate use ATS and RIS when you define the replicate by including the **--ats** and **--ris** options in the replicate definition.

Tip: Until you become thoroughly familiar with the behavior of the replication system, select both ATS and RIS options.

Related concepts:

Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1

Related tasks:

"Replicating Only Changed Columns"

"Creating ATS and RIS Directories" on page 4-13

"Defining Replication Servers" on page 6-1

Related reference:

"cdr define replicate" on page A-61

Replicating Only Changed Columns

You can choose to replicate only those columns that have changes instead of entire rows.

By default, Enterprise Replication replicates the entire row, even if only one column changed. Exceptions to this are columns containing BLOB or CLOB data: these columns are updated *only* when their contents have changed, regardless of whether other columns have changed or not. This prevents large amounts of data from being unnecessarily transmitted. Enterprise Replication always sends the primary key column, even if you specify to replicate only changed columns.

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.

Replicating only the columns that changed has the following advantages:

- Sends less data, as only the data that needs to be modified is sent
- Uses less 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 (unless you are using the delete wins conflict resolution rule). By replicating the entire row, Enterprise Replication 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.

Replicating only the columns that changed has the following disadvantages:

- 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, Enterprise Replication logs this information as a failed operation.
- You cannot use the SPL routine or time stamp with SPL routine conflict-resolution rules.
- You cannot use update-anywhere replication; doing so can result in inconsistent conflict resolution.
- All database servers in the domain 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 corrupt the 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 *IBM Informix Administrator's Reference*.

Related concepts:

"Conflict Resolution" on page 3-6

"Disk Space for Delete Tables" on page 4-8

"Considerations for Replicating Opaque Data Types" on page 2-17

Related tasks:

"Setting Up Failed Transaction Logging" on page 6-11

Related reference:

"cdr define replicate" on page A-61

Using the IEEE Floating Point or Canonical Format

You can specify how the FLOAT and SMALLFLOAT data types are handled, depending on your platform.

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 IEEE floating point format, include the **--floatieee** option in your replicate definition.

  It is recommended 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; it is recommended 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-65.

Important: You cannot modify the replicate to change the **--floatieee** or **--floatcanon** options.

Related reference:

"cdr define replicate" on page A-61

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.

When you design your triggers, you can use the **'cdrsession'** option of the **DBINFO()** function to determine if the transaction is a replicated transaction.

For information, refer to “Triggers” on page 2-8 and “Special Options” on page A-65.

Related reference:

“cdr define replicate” on page A-61

Enabling code set conversion between replicates

You can enable code set conversion to allow replication of data between servers that use different code sets.

Prerequisites:

The table and column names must contain ASCII characters in order to convert a non-master replicate to a master replicate.

The servers must have UTF-8 code set transaction support enabled in order to replicate between server versions.

The target schema should allow for expansion due to codeset conversion. For example, a CHAR(10) column in one codeset might require 40 bytes in the converted codeset.

When code set conversion is enabled, character columns of the following data types are converted to UTF-8 (Unicode) when the row is copied into the transmission queue.

- CHAR
- VARCHAR
- NCHAR
- NVARCHAR
- LVARCHAR
- TEXT
- CLOB

When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set that is used on the target server. No attempt is made to convert character data contained within opaque data types such as user-defined types (UDTs) or DataBlade module data.

To enable code set conversion between replicates, include the **--UTF8=y** option in your replicate definition.

To use the latest version of the Unicode library, set the GL_USEGLU environment variable in your server environment.

If your table names or column names contain non-ASCII characters, you must manually create a shadow replicate and then swap the shadow replicate with the primary replicate using the **cdr swap shadow** command.

The autocreate option is not supported for replicates defined with **--UTF8=y** option when using the **cdr realize template** or **cdr change replicate** commands.

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

For additional information, refer to "Special Options" on page A-65.

Related concepts:

"Using GLS with Enterprise Replication" on page 2-13

Related reference:

"cdr swap shadow" on page A-159

Configuring code set conversion between replicates

The examples in this topic show how to create replicate and template definitions while replicating data between databases that use different code sets.

When non-English characters are used for database, table, column, or owner names, each server must be added to the UTF-8 realize template definition by connecting to the server locally. Only one server at a time should be added to the replicate definition using the **change replicate** command. You cannot add multiple servers to a replication definition using the **define repl** command unless the database code set number is the same for all servers. The CLIENT_LOCALE environment variable must be set unless the database locale is en_us.819. Replicate and template names must be in English.

This example shows how to create and realize a template on two servers, named *node_1* and *node_2*. For this example, assume that *node_1* uses **de_de.819** locale and *node_2* uses **de_de.utf8** locale:

1. On node_1, run the following commands:

```
export DB_LOCALE=de_de.819
export CLIENT_LOCALE=de_de.819
cdr define template set1 -C always -M g_node1 -S row -d testdb -a -A -R --UTF8=y
cdr realize template set1 g_node1
```

2. On node_1 run the following command and wait for the **Txns in queue** count to go to zero.

```
onstat -g rqm cntrlq
```

3. On node_2, run the following commands:

```
export DB_LOCALE=de_de.utf8
export CLIENT_LOCALE=de_de.utf8
cdr realize template set1 g_node2
```

The following steps show how to define a replicate when non-ASCII characters are used for table, column, owner, or database names. Before starting, ensure that the replicate name uses English ASCII characters and that the DB_LOCALE environment variable on the server is set to the same value as the locale of the participant being added.

1. Define the replicate with the first participant and then connect to the participant.
2. Add and connect to each additional participant, one participant at a time.
3. When all of the participants have been added, ensure that the control queue is empty and start the replicate definition.

You can check the control queue message count using the **onstat -g rqm cntrlq**. Wait for the **Txns in queue** count to go zero.

The following example shows how to create a replicate definition between two servers to replicate data between de_de.819 and de_de.utf8 databases:

1. On server node_1, run the following commands:

```
export DB_LOCALE=de_de.819
export CLIENT_LOCALE=de_de.819
cdr define repl german_repl  -M g_node1 -C always -S transaction
         -A -R -I --UTF8=y "testdb@g_node1:user1.table1" "select * from table1"
```

2. On node_1 run the following command and wait for the **Txns in queue** count to go to zero.

```
onstat -g rqm cntrlq
```

3. On node_2, run the following commands:

```
export DB_LOCALE=de_de.utf8
export CLIENT_LOCALE=de_de.utf8
cdr change repl -c node2 -a german_repl
           "testdb@g_node2:user1.table1" "select * from table1"
```

4. On node_2 run the following command and wait for the **Txns in queue** count to go to zero.

```
onstat -g rqm cntrlq
```

5. Run the following command on either server:

```
cdr start repl german_repl
```

Code set conversion errors

You can use the ATS and RIS files to identify problems that occur during code set conversion.

To specify which warnings and errors to suppress, use the CDR_SUPPRESS_ATSRISWARN configuration parameter. For more information, see "CDR_SUPPRESS_ATSRISWARN Configuration Parameter" on page B-14

Each column in the RIS file begins with (W) if substitute characters were added to the column data or (E) if data was rejected because of a UTF-8 conversion failure.

Examples of conversion errors:

On the source server, a row of data fails conversion to UTF-8 code set.

Data sync error 63 is stored in an RIS file on the source server. The RIS file contains the row that failed to convert; the failed row is not converted and is not replicated on the target server. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:0 / Name:*UNKNOWN* / CommitTime:
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:35:22
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:63 (Error while converting data from local database codeset to
    UTF8.) / SQL:0 / ISAM:0
LRH  Failed column list: charcol (W), ncharcol (E)
LRD 3|Lkqy|jvdHj@ifcjuWg|biLs|uk|RwvCZOpfpqruLAA|JloY|<27, TEXT,
    PB 1 (utm_group_1) 1305051204 (11/05/10 13:13:24)>|<4, TEXT, BB>|
    <18, CLOB, SB 1305051204 (11/05/10 13:13:24)>
RRD ||||||||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

On the source server, conversion from the local code set to UTF-8 resulted in the substitution of one or more characters in the row.

Data sync error 65 is stored in an RIS file on the source server, and the row is replicated. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:0 / Name:*UNKNOWN* / CommitTime:
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:32:14
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:65 (Substitute characters added  while converting data from local
    database codeset to UTF8.) / SQL:0 / ISAM:0
LRH Failed column list: charcol (W), ncharcol (W), vchar (W), nvchar (W),
    lvchar (W)
LRD 2|iU\VoJMZ|axhGRxKmDW|e@Xv|biLs|pyqasjUpAc{wCu|efM@}Vd|<22, TEXT,
    PB 1 (utm_group_1) 1305051204 (11/05/10 13:13:24)>|<36, TEXT, BB>
    |<15, CLOB, SB 1305051204 (11/05/10 13:13:24)>
RRD ||||||||
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

On the target server, a row of data failed to convert from UTF-8 format to the local database code set.

Data sync error 64 is stored in an ATS/RIS file on the target server, and the row or transaction is aborted depending on the replicate scope. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:1 / Name:utm_group_1 / CommitTime:11-05-10 13:40:19
TXH Target ID:3 / Name:utm_group_3 / ReceiveTime:11-05-10 13:40:19
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:64 (Error while converting data from UTF8 to local database
    codeset.) / SQL:0 / ISAM:0
RRH  Failed column list: vchar (E)
RRD 3||jdicW|?|?|?|?|?|?
==========
TXH Transaction aborted
TXH ATS file:/usr4/nagaraju/utm/tmp/ats.utm_group_3.utm_group_1.D_3
    .110510_13:40:19.2 has also been created for this transaction
```

On the target server, conversion from UTF-8 to the local server code set resulted in the substitution of one or more characters in the row.

Data sync error 66 is stored in a warning RIS file on the target server, and the row is applied. A list of column names that failed to convert is also stored in the RIS file. Example RIS file:

```
TXH Source ID:3 / Name:utm_group_3 / CommitTime:11-05-10 13:13:58
TXH Target ID:1 / Name:utm_group_1 / ReceiveTime:11-05-10 13:13:58
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Insert
RRH CDR:66 (Substitute characters added while converting data
    from UTF8 to local database codeset.) / SQL:0 / ISAM:0
RRH  Failed column list: charcol (W), ncharcol (W), vchar (W),
    nvchar (W), lvchar (W), textcol (W), textbcol (W), clobcol (W)
RRD 99||keI||m||<46, TEXT, PB 3 (utm_group_3) 1305051238
    (11/05/10 13:13:58)>|<68, TEXT, BB>|<13, CLOB, SB>
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

Text and CLOB data conversion failures

If the conversion of text or CLOB data to UTF-8 fails on the source server then the blob buffer is marked with the appropriate error and the target

servers create ATS/RIS files for these blob data conversion failures. Example text column conversion error:

```
TXH Source ID:1 / Name:utm_group_1 / CommitTime:11-05-10 12:26:30
TXH Target ID:3 / Name:utm_group_3 / ReceiveTime:11-05-10 12:28:15
----------
RRH Row:1 / Replicate Id: 65540 / Table: testdb@usr1.utf8tab / DbOp:Update
RRH CDR:65 (Substitute characters added  while converting data from local
    database codeset to UTF8.) / SQL:0 / ISAM:0
RRH  Failed column list: textcol (W)
LRD 2|<46, TEXT, PB 1 (utm_group_1) 1305048215 (11/05/10 12:23:35)
    >|<40, CLOB, SB 1305048215 (11/05/10 12:23:35)>
RRD 2|<44, TEXT, PB 1 (utm_group_1) 1305048390 (11/05/10
    12:26:30)>|<0(NoChange), CLOB, SB>
==========
TXH Transaction committed
TXH Total number of rows in transaction:1
```

Defining 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*.

Related reference:

"cdr define replicateset" on page A-69

Exclusive Replicate Sets

If your replicated tables use referential integrity and are defined with time-based replication, you must create an *exclusive replicate set*. If your replicates use referential integrity and you plan to stop and start the replicate set, you should also use an exclusive replicate set.

An exclusive replicate set has the following characteristics:

- All replicates in an exclusive replicate set have the same state and frequency settings. For more information, see "cdr list replicateset" on page A-101.
- 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" on page 8-7
  - "Stopping a Replicate" on page 8-8
  - "Suspending a Replicate" on page 8-8
  - "Resuming a Suspended Replicate" on page 8-9
- Replicates that belong to an exclusive replicate set cannot belong to any other replicate sets.

To create an exclusive replicate set, use the **--exclusive** option with **cdr define replicateset**.

Important: You cannot change an exclusive replicate set to non-exclusive.

Related reference:

"cdr define replicateset" on page A-69

"cdr define template" on page A-75

"cdr resume replicate" on page A-126

Non-Exclusive Replicate Sets

By default, the **cdr define replicateset** command 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: You cannot change a non-exclusive replicate set 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 6-11) 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 individually for replicates in the set.

For more information, see "cdr define replicateset" on page A-69.

Initially Synchronizing Data Among Database Servers

Enterprise Replication provides an initial synchronization feature that allows you to easily bring a new table up-to-date with replication when you start a new replicate, or when you add a new participant to an existing replicate.

You do not need to suspend any servers that are replicating data while you add the new replicate and synchronize it.

The **cdr start replicate** and **cdr start replicateset** commands provide options to perform an initial synchronization for the replicates you are starting. All of the rows that match the replication criteria will be transferred from the source server to the target servers. If you are starting a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables).

Use the **--syncdatasource** (**-S**) option of the **cdr start replicate** or **cdr start replicateset** command to specify the source server for synchronization. Any existing rows in the specified replicates are deleted from the remote tables and replaced by the data from the node you specify using **-S**.

The **--extratargetrows** option of the **cdr start replicate** or **cdr start replicateset** commands specifies how to handle rows found on the target servers that are not present on the source server. You can specify to remove rows from the target, keep extra rows on the target, or replicate extra rows from the target to other participants.

If you use the **cdr start replicate** or **cdr start replicateset** command to specify a subset of servers on which to start the replicate (or replicate set), that replicate (or replicate set) must already be active on the source server. The source server is the server you specify with the **-S** option. For example, for the following command, **repl1** must already be active on **serv1**:

```
cdr start repl repl1 ... -S serv1 serv2 serv3
```

When you start a replicate (or replicate set) for participants on all servers, the replicate does not need to be active on the source server. So, for the following command, **repl1** does not need to be active:

```
cdr start repl1 ... -S serv1
```

When Enterprise Replication performs initial data synchronization, it keeps track of discrepancies between the constraints set up on source and target server tables. Rows that fail to be repaired due to these discrepancies are recorded in the ATS and RIS files.

If replication fails for some reason and data becomes inconsistent, there are different ways to correct data mismatches between replicated tables while replication is active. The recommended method is direct synchronization. You can also repair data based on an ATS or RIS file. Both of these methods are described in "Resynchronizing Data among Replication Servers" on page 8-14.

Related concepts:

"Repair and Initial Data Synchronization" on page 1-3

"Load and unload data" on page 4-24

Using Templates to Set Up Replication

Enterprise Replication provides templates to allow easy set up and deployment of replication for clients with large numbers of tables to replicate. A template uses schema information about a database, a group of tables, columns, and primary keys to define a group of master replicates and a replicate set.

First, you create a template using the **cdr define template** command; then, you instantiate it on the servers where you want to replicate data by running the **cdr realize template** command.

Important: If you want to use time-based replication, you must set up replication manually.

Important: Templates set up replication for full rows of tables (all the columns in the table), because they are designed to facilitate setting up large scale replication environments. If you want a participant to contain a partial row (just some columns in the table), you can either set up replication manually, or, after you realize a template you can use the **cdr remaster** command to restrict the query.

All options of the **cdr define template**, **cdr list template**, **cdr realize template**, and **cdr delete template** commands are described in detail in Appendix A, "The cdr Command-Line Utility Reference," on page A-1.

Defining Templates

You define a template using the **cdr define template** command, with which you can specify which tables to use, the database and server they are located in, and whether to create an exclusive or non-exclusive replicate set. Table names can be listed on the command line or accessed from a file using the **--file** option, or all tables in a database can be selected.

Important: A template cannot define tables from more than one database.

Specify that the replicate set is exclusive if you have referential constraints on the replicated columns. Also, if you create an exclusive replicate set using a template, you do not need to stop the replicate set to add replicates. For more information about exclusive replicate sets, see "Defining Replicate Sets" on page 6-18.

A template defines a group of master replicates and a replicate set.

You can use the **cdr list template** command from a non-leaf node to view details about the template, including the internally generated names of the master replicates. These are unique names based on the template, the server, and table names.

Realizing Templates

After you define a template using the **cdr define template** command, use the **cdr realize template** command to instantiate the template on your Enterprise

Replication database servers. The **cdr realize template** command first verifies that the tables on each node match the master definition used to create the template. Then, on each node, it adds the tables defined in the template as participants to master replicates created by the template.

If a table on a server has additional columns to those defined in the template, those columns are not considered part of the replicate.

If a table does not already exist on a server where you realize the template, you can choose to create it, and it is also added to the replicate.

Also, at realization time, you can also choose to synchronize data among all servers.

Verifying Participants without Applying the Template

The **--verify** option allows you to check that a template's schema information is correct on all servers before actually instantiating the template.

Synchronizing Data Among Database Servers

Use the **--syncdatasource** option to specify a server to act as the source for data synchronization on all servers where you are realizing the template. The server listed with this option must either be listed as one of the servers on which to realize the template, or it must already have the template.

Improve Performance During Synchronization:

You can speed up a synchronization operation by temporarily increasing the size of the send queue.

Enterprise Replication uses the value of the CDR_QUEUEMEM configuration parameter as the size of the send queue during a synchronization operation. To increase the size of the send queue during a particular synchronization operation, use the **--memadjust** option.

In addition to controlling memory during initial synchronization, you can also control memory consumption when you realize a template and perform a direct synchronization.

Creating Tables Automatically

The **--autocreate** option allows you to choose to automatically create tables in the template definition if they do not already exist on a server. (This cannot be done for tables that contain user-defined data types.)

Use the **--dbspace** option to specify a dbspace for table creation.

Note: Tables created with autocreate do not automatically include non-primary key indices, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with autocreate require the use of these objects you must explicitly create the objects by hand.

Other synchronization options

Several other options to the **cdr realize template** command can affect how synchronization occurs.

You can use the **--applyasowner** option to realize a table by its owner rather than the user **informix**.

The **--extratargetrows** option specifies whether to delete, keep, or merge rows found on target servers that are not present on the source server during the synchronization operation.

The **--target** option defines whether target servers are receive-only (when target servers are defined as receive-only, updates made on those servers are not propagated).

Changing Templates

You cannot update a template. To adjust a template, you must delete it with the **cdr delete template** command and then re-create it with the **cdr define template** command.

Template Example

This example illustrates a scenario in which one template is created, and then a second template is added and realized on the same servers. The replicates in both templates are consolidated into the first template for ease of maintenance, and the second template is then deleted.

The first template **Replicateset1** is defined on three tables in the **college** database: **staff**, **students**, and **schedule**. The template is realized on the servers **g_cdr_ol_1** and **g_cdr_ol_2**.

The second template **Replicateset2** is defined on three tables in the **bank** database: **account**, **teller**, and **transaction**. This template is realized on the same servers as the first template: **g_cdr_ol_1** and **g_cdr_ol_2**.

The replicates in both templates exist on the same servers, and would be administered (for example, stopped and started) at the same time. Thus, the replicates defined as part of **Replicateset2** can be moved into **Replicateset1**, after which the **Replicateset2** template can then be deleted.

This procedure is performed as follows:

1. Define the template **Replicateset1** on the **staff**, **students**, and **schedule** tables of the **college** database:

   ```
   cdr define template -c g_cdr_ol_1 Replicateset1 -M g_cdr_ol_1\
    -C "timestamp" -A -R -d college testadm.staff testadm.students\
    testadm.schedule
   ```

   This command also creates the replicate set **Replicateset1**.
2. Realize the template on the server **g_cdr_ol_1**:

   ```
   cdr realize template -c g_cdr_ol_1 Replicateset1 "college@g_cdr_ol_1"
   ```
3. Realize the template on server **g_cdr_ol_2** and synchronize the data with server **g_cdr_ol_1**:

   ```
   cdr realize template -c g_cdr_ol_2 -u -S g_cdr_ol_1 \
   Replicateset1 "university@g_cdr_ol_2"
   ```
4. Define the template **Replicateset2** on the **account**, **teller**, **transaction**, and **customer** tables of the **bank** database:

   ```
   cdr define template -c g_cdr_qa_1 Replicateset2 -M g_cdr_ol_1\
      -C "timestamp" -A -R -d bank testadm.account testadm.teller\
     testadm.transactions testadm.customer
   Obtaining dictionary for bank@g_cdr_ol_1:'testadm'.account
   ```

```
Obtaining dictionary for bank@g_cdr_ol_1:'testadm'.teller
Obtaining dictionary for bank@g_cdr_ol_1:'testadm'.transactions
Obtaining dictionary for bank@g_cdr_ol_1:'testadm'.customer
Creating mastered replicate Replicateset2_g_cdr_ol_1_1_1_account
  for table 'testadm'.account
Creating mastered replicate Replicateset2_g_cdr_ol_1_1_2_teller
  for table 'testadm'.teller
Creating mastered replicate Replicateset2_g_cdr_ol_1_1_3_transactions
  for table 'testadm'.transactions
Creating mastered replicate Replicateset2_g_cdr_ol_1_1_4_customer
  for table 'testadm'.customer
```

This command also creates the replicate set **Replicateset2**.

5. Realize the template **Replicateset2** on **g_cdr_ol_1**:

```
cdr realize template -c g_cdr_ol_1 Replicateset2 "bank@g_cdr_ol_1"
```

6. Realize the template on server **g_cdr_ol_2** and synchronize the data with server **g_cdr_ol_1**:

```
cdr realize template -c g_cdr_ol_1 -u -S g_cdr_ol_1 \
Replicateset2 "bank@g_cdr_ol_2"
```

7. Add the replicates created as part of **Replicateset2** to **Replicateset1**. (Use the **cdr list replset Replicateset2** command to list the replicates in **Replicateset2**.):

```
cdr change replset -c g_cdr_ol_1 -a Replicateset1\
  Replicateset2_g_cdr_ol_1_1_1_account \
Replicateset2_g_cdr_ol_1_1_2_teller \
 Replicateset2_g_cdr_ol_1_1_3_transactions \
Replicateset2_g_cdr_ol_1_1_4_customer
```

8. Delete the replicate set **Replicateset2**:

```
cdr delete template Replicateset2
```

9. Realize all the replicates on a new server **g_cdr_ol_3**. Then realize the template **Replicateset1** on the server **g_cdr_ol_3**:

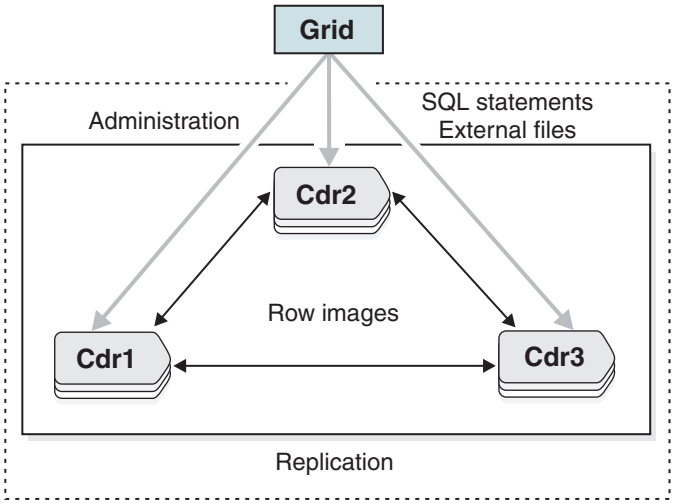
```
cdr realize template -c g_cdr_ol_1 -u -S g_cdr_ol_1\
  Replicateset1 "bank@g_cdr_ol_3"
```

This command adds **g_cdr_ol_3** as a participant to all the replicates in **Replicateset1**, including the replicates that were created as part of the template **Replicateset2**: **Replicateset2_g_cdr_ol_1_1_1_account**, **Replicateset2_g_cdr_ol_1_1_2_teller**, **Replicateset2_g_cdr_ol_1_1_3_transactions**, and **Replicateset2_g_cdr_ol_1_1_4_customer**.

Chapter 7. Grid setup and management

A grid is a set of replication servers to simplify administration. When you run SQL data definition statements from within a grid context on one server in the grid, they are propagated to all other servers in the grid. You can run SQL data manipulation statements and routines through grid routines. You can choose to set up replication automatically when you create a table through a grid. You can propagate external files to other servers in the grid.

In contrast, SQL statements are not replicated by Enterprise Replication. Enterprise Replication replicates the row images that are the results from SQL statements. The grid propagates SQL statements, but does not, by default, propagate the results of propagated SQL statements. The following illustration shows three replication servers, named **Cdr1**, **Cdr2**, and **Cdr3**, that replicate row images between each other, while the grid propagates SQL statements and administration commands.



A grid can be useful if you have multiple replication servers and you often need to perform the same tasks on every replication server. The following types of tasks are easily run through the grid:

- Administering servers, for example, adding chunks, removing logical logs, or changing configuration parameter settings
- Updating the database schema, for example, altering tables or adding tables
- Running or creating stored procedures or user-defined routines
- Updating data, for example, purging old data or updating values based on conditions
- Enabling replication when creating a table
- Altering a replicate definition when altering a replicated table
- Copying external files to grid servers

For example, suppose that you have 100 replication servers and need to create a table. You need to fragment the table into two new dbspaces. You also need to create a new stored procedure to run on the table. With a grid, you would run four commands to perform these tasks on all 100 replication servers, instead of running 400 commands. The command to create the table can also specify that the data in that table is replicated.

You can control the security of the grid by authorizing which users can run grid routines on which servers. You can monitor the results of grid routines and rerun any failed routines on the appropriate servers.

You can route client connections to servers in the grid based on the quality of replicated data and transaction latency by configuring service-level agreements for a Connection Manager.

Example of setting up a replication system with a grid

This comprehensive example sets up a replication domain, creating a grid, creating a database, creating a replicated table, and loading data.

This example creates a replication domain and grid that contain four replication servers: **serv1**, **serv2**, **serv3**, **serv4**. Each server computer has the Informix database server installed, but no databases defined.

1. On all servers, set the CDR_QDATA_SBSPACE configuration parameter.
2. Edit the `sqlhosts` files on all four servers so that they each have the following information:

```
gserv1     group    -                      -     i=143
serv1      ontlitcp ny.usa.com             1230  g=gserv1
gserv2     group    -                      -     i=144
serv2      ontlitcp tokyo.japan.com        1231  g=gserv2
gserv3     group    -                      -     i=145
serv3      ontlitcp rome.italy.com         1232  g=gserv3
gserv4     group    -                      -     i=146
serv4      ontlitcp perth.australia.com    1233  g=gserv4
```

3. Define each server as a replication server by running the **cdr define server** command:

```
cdr define server -c gserv1 -I gserv1
cdr define server -c gserv2 -S gserv1 -I gserv2
cdr define server -c gserv3 -S gserv1 -I gserv3
cdr define server -c gserv4 -S gserv1 -I gserv4
```

4. Create a grid that includes all replication servers in the domain as members of the grid:

```
cdr define grid grid1 --all
```

5. Authorize the user **bill** to run commands on the grid and designate the server **gserv1** as the source server from which grid commands can be run:

```
cdr enable grid --grid=grid1 --user=bill --node=gserv1
```

   Tip: User **informix** does not have permission to run grid operations unless you include it in the user list.

6. Run **cdr list grid** to see the grid configuration:

```
Grid              Node              User
----------------- ----------------- ----------------
grid1             gserv1*           bill
                  gserv2
                  gserv3
                  gserv4
```

   The asterisk indicates that **gserv1** is the source server for the grid.

7. Run the **cdr list replicateset** command to see the grid replicate set information:

```
Ex T REPLSET                 PARTICIPANTS
---------------------------------------------
Y  Y  grid1
```

The replicate set has the same name as the grid. It does not yet contain any participants.

8. Create two dbspaces named **dbsp2** and **dbsp3** in which to fragment a table:

```
database sysmaster;

EXECUTE FUNCTION ifx_grid_function('grid1',
  'task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0")');

EXECUTE FUNCTION ifx_grid_function('grid1',
  'task("create dbspace","dbsp3",
          "/db/chunks/dbsp3","8G","0")');
```

The dbspaces are created on all four servers.

9. Create database named **retail** and a table named **special_offers** with replication enabled:

```
database sysmaster;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE DATABASE retail WITH LOG;

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar,
    offer_type        char(16))
    WITH CRCOLS
FRAGMENT BY EXPRESSION
    offer_type = "GOLD" IN dbsp2,
    REMAINDER IN dbsp3;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

10. Run the **cdr list grid --verbose grid1** command to see information about the statements on each server:

```
Grid              Node              User
----------------- ----------------- ----------------
grid1             gserv1*           bill
                  gserv2
                  gserv3
                  gserv4
Details for grid grid1

Node:gserv1 Stmtid:1 User:bill Database:retail 2010-05-27 15:21:57
CREATE DATABASE retail WITH LOG;
ACK gserv1 2010-05-27 15:21:57
ACK gserv2 2010-05-27 15:21:58
ACK gserv3 2010-05-27 15:21:59
ACK gserv4 2010-05-27 15:21:59

Node:gserv1 Stmtid:1 User:bill Database:retail 2010-05-27 15:21:57
CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar
    offer_type        char(16))
    WITH CRCOLS
FRAGMENT BY EXPRESSION
    offer_type = "GOLD" IN dbsp2
    REMAINDER IN dbsp3;
```

```
ACK gserv1 2010-05-27 15:21:57
ACK gserv2 2010-05-27 15:21:58
ACK gserv3 2010-05-27 15:21:59
ACK gserv4 2010-05-27 15:21:59
```

Both statements succeeded on all four servers.

11. Run **cdr list replicate** to see the replicate information:

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------
REPLICATE:        gserv1_1
STATE:            Active
CONFLICT:         Timestamp
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      retail:bill.special_offers
OPTIONS:
REPLTYPE:         Master,Grid
```

The replicate was created and is active.

12. Run the **cdr list replicate brief gserv1_1** command to see the participants:

```
REPLICATE  TABLE                              SELECT
-----------------------------------------------------------------
gserv1_1   retail@gserv1:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
gserv1_1   retail@gserv2:bill.special_offers  select * from
                                               bill.special_offers
```

13. Load data onto one of the replication servers and Enterprise Replication replicates the data to the other servers. For more information, see "Load and unload data" on page 4-24.

Related tasks:

"Adding a server to a grid by cloning" on page 7-5

"Routing client connections in a grid" on page 7-16

Related reference:

"cdr enable grid" on page A-89

"cdr list grid" on page A-94

"cdr list replicateset" on page A-101

Creating a grid

You can create a grid based on an existing replication domain. You must authorize users who can run grid routines, and designate a server from which to run grid routines.

You must be connected to a replication server in the domain that contains the servers that you want to include in the grid.

To create a grid:

1. Specify a name for the grid and the servers to include in the grid by running the **cdr define grid** command. For example, the following command creates a grid named **grid1** and adds all replication servers in the domain as members of the grid:

```
cdr define grid grid1 --all
```

2. Authorize users to run commands on the grid and designate a server from which grid commands can be run by running the **cdr enable grid** command. For example, the following command authorizes the user **bill** to run commands on the server **gserv1**:

   ```
   cdr enable grid --grid=grid1 --user=bill --node=gserv1
   ```

   Only authorized users can run grid routines on authorized servers. User **informix** does not have permission to perform grid operations unless you include it in the user list.

Related reference:

"cdr define grid" on page A-58

"cdr enable grid" on page A-89

Maintaining the grid

You can change which servers are members of the grid. You can change which users are authorized to run grid routines and change from which servers grid routines can be run.

If you add a server to your replication domain, you might want to add it to the grid. If you remove a server from your replication domain, you should remove it from the grid.

To add or remove replication servers from the grid, run the **cdr change grid** command. For example, to add a server named **gserv3** to the grid **grid1**, run this command:

```
cdr change grid grid1 --add=gserv3
```

To change which users can run routines on the grid or which servers are authorized to run grid routines, run the **cdr enable grid** and **cdr disable grid** commands. For example, to change the authorized server from **gserv1** to **gserv2** and authorize the user **srini**, run the following commands:

```
cdr disable grid --grid=grid1 --node=gserv1
cdr enable grid --grid=grid1 --node=gserv2 --user=srini
```

To see information about the grid, such as, which servers can run grid routines and the status of routines run on the grid servers, run the **cdr list grid** command.

To delete the history of grid routines, run the **ifx_grid_purge()** procedure. You must occasionally purge information about completed grid routines to prevent the **syscdr** database from growing too large.

To delete a grid, use the **cdr delete grid** command.

Related reference:

"cdr change grid" on page A-30

"cdr disable grid" on page A-85

"cdr enable grid" on page A-89

"cdr list grid" on page A-94

"ifx_grid_purge() procedure" on page C-7

Adding a server to a grid by cloning

You can add a new server to a grid by cloning an existing replication server in the grid.

The server you are adding to the grid must have the same hardware and operating system as the source server that you are cloning.

To add a server to a grid:

Clone an existing replication server in the grid by using the **ifxclone** utility with the **--disposition=ER** option. This process is described in "Adding a server to the domain by cloning a server" on page 6-5.

The following example adds a fifth server, named **serv5**, to an existing replication domain and to a grid named **grid1**. The server **serv1** is used as the source server.

1. On the **serv1** server, set the value of the ENABLE_SNAPSHOT_CLONE configuration parameter to 1 in the `onconfig` file.
2. On the **serv5** servers, complete the **ifxclone** prerequisites for all servers, such as setting the required configuration parameters and environment variables.

   Set these environment variables:

   - **INFORMIXDIR**
   - **INFORMIXSERVER**
   - **INFORMIXSQLHOSTS**
   - **ONCONFIG**

   Set these configuration parameters to the same values on the **serv5** server as on the **serv1** server:

   - DRAUTO
   - DRINTERVAL
   - DRTIMEOUT
   - LOGBUFF
   - LOGFILES
   - LOGSIZE
   - LTAPEBLK
   - LTAPESIZE
   - ROOTNAME
   - ROOTSIZE
   - PHYSBUFF
   - PHYSFILE
   - STACKSIZE
   - TAPEBLK
   - TAPESIZE
3. On the **serv5** server, create all of the chunks that exist on the **serv1** server.
4. On the **serv5** server, run the **ifxclone** command with the **--disposition=ER** option to clone the data and the configuration of the **serv1** server onto the **serv5** server:

   ```
   ifxclone --trusted --source=serv1 --sourceIP=111.222.333.444
   --sourcePort=1230 --target=serv5 --targetIP=111.222.333.777
   --targetPort=1234 --disposition=ER
   ```
5. Edit the `sqlhosts` files on all five servers in the domain so that they each have the following information:

   ```
   gserv1     group    -                    -     i=143
   serv1      ontlitcp ny.usa.com           1230  g=gserv1
   gserv2     group    -                    -     i=144
   serv2      ontlitcp tokyo.japan.com      1231  g=gserv2
   ```

```
gserv3     group    -                      -     i=145
serv3      ontlitcp rome.italy.com         1232  g=gserv3
gserv4     group    -                      -     i=146
serv4      ontlitcp perth.australia.com    1233  g=gserv4
gserv5     group    -                      -     i=147
serv5      ontlitcp helsinki.finland.com   1234  g=gserv5
```

The server **serv5** is automatically added to the grid **grid1**.

Related concepts:

"Example of setting up a replication system with a grid" on page 7-2

Related tasks:

"Adding a server to the domain by cloning a server" on page 6-5

Related reference:

"cdr change grid" on page A-30

Adding an existing replicate to a grid replicate set

You can add replicates created outside of a grid environment to a grid replicate set by using the **cdr change replicateset** command. Replicated tables that are altered through the grid are added to the grid replicate set automatically.

A grid replicate set must already exist.

An existing replicate must have the following properties for it to be added to a grid replicate set:

- Its participant servers must be the same as the servers in the grid.
- The replicated table schema must be the same among all participants.
- The entire replicated table is replicated. Using a SELECT statement in the participant definition that does not include all the columns in the table or includes a WHERE clause is not allowed.

To add a replicate to a grid replicate set:

Run the **cdr change replicateset** command with the **--add** option and specifying the grid replicate set. For example, the following command adds a replicate named **vendors** to the **grid1** grid replicate set:

```
cdr change replicateset --add grid1 vendors
```

When you run the **cdr list replicate** command, the `REPLTYPE` field shows `Grid`.

Related tasks:

"Altering replicated tables through a grid" on page 7-11

Related reference:

"cdr change replicateset" on page A-35

"cdr list replicate" on page A-97

Viewing grid information

You can view information about a grid and whether a replicate or replicate set belongs to a grid.

To view information about a grid:

Run the **cdr list grid** command. For example, the following command shows the servers and authorized users for a grid named **grid1**:

```
cdr list grid grid1
```

The output for this command might be:

```
Grid                 Node                 User
-------------------- -------------------- ----------------
grid1                gserv1*              bill
                     gserv2
                     gserv3
                     gserv4
```

The user **bill** is authorized to run grid commands on the server **gserv1**.

You can see whether a replicate is a member of a grid replicate set by running the **cdr list replicate** command or the **onstat -g cat repls** command. You can also query the **syscdrrepl** SMI table. The following example output of the **cdr list replicate** command shows that the replicate is a master replicate and a member of a grid replicate set:

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------
REPLICATE:        grid_6553604_100_3
STATE:            Active ON:g_delhi
CONFLICT:         Always Apply
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      tdb:nagaraju.t1
OPTIONS:          row,ris,fullrow
REPLID:           6553605 / 0x640005
REPLMODE:         PRIMARY  ON:gserv1
APPLY-AS:         INFORMIX ON:gserv1
REPLTYPE:         Master,Grid
```

Related reference:

"cdr list replicateset" on page A-101

"cdr list replicate" on page A-97

"onstat -g cat" on page E-2

"The syscdrrepl Table" on page G-12

Administering servers in the grid with the SQL administration API

You can run SQL administration API commands in grid routines to perform administrative tasks on all servers in the grid.

The grid must exist and you must run the grid routines as an authorized user from an authorized server and while connected to the **sysadmin** database.

To propagate an SQL administration API command:

1. Run the **ifx_grid_function()** function with the SQL administration API command as the second argument.
2. Check the return code of the SQL administration API command to determine if it succeeded by running the **cdr list grid** command. The **cdr list grid** command shows the return code. The status of the **ifx_grid_function()** function can be ACK, which indicates success, even if the SQL administration API command failed.

Examples

The following examples must be run in the **sysadmin** database.

Example 1: Change a configuration parameter setting

The following example sets the maximum size of the log staging directory to 100 KB on all the servers in the grid:

```
EXECUTE FUNCTION ifx_grid_function('grid1',
  'admin("set onconfig permanent",
          "CDR_LOG_STAGING_MAXSIZE","100")');
```

The output of the **cdr list grid** command shows that the **admin()** function succeeded because the return codes are positive numbers:

```
Grid                Node                User
------------------ ------------------ ----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
admin("set onconfig permanent",
          "CDR_LOG_STAGING_MAXSIZE","100")
ACK cdr1 2010-05-27 15:21:57
    '110'
ACK cdr2 2010-05-27 15:21:58
    '111'
ACK cdr3 2010-05-27 15:21:58
    '112'
```

Example 2: Create a new dbspace

The following example creates a new dbspace on all the servers in the **grid1** grid:

```
EXECUTE FUNCTION ifx_grid_function('grid1',
  'task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0")');
```

The output of the **cdr list grid** command shows that the **task()** function failed:

```
Grid                Node                User
------------------ ------------------ ----------------
grid1               cdr1*               bill
                    cdr2
                    cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
task("create dbspace","dbsp2",
          "/db/chunks/dbsp2","2G","0"
ACK cdr1 2010-05-27 15:21:57
    'Unable to create file /db/chunks/dbsp2'
ACK cdr2 2010-05-27 15:21:58
    'Unable to create file /db/chunks/dbsp2'
ACK cdr3 2010-05-27 15:21:58
    'Unable to create file /db/chunks/dbsp2'
```

Related reference:

"ifx_grid_function() function" on page C-6

"ifx_grid_execute() procedure" on page C-4

Propagating database object changes

You can make changes to database objects while connected to the grid and propagate the changes to all the servers in the grid.

You can propagate creating, altering, and dropping database objects to servers in the grid. For example, you can create a new database or table or change an existing database or table. You can also create stored procedures and user-defined routines.

The grid must exist and you must run the grid routines as an authorized user from an authorized server.

To propagate database object changes:

1. Connect to the grid by running the **ifx_grid_connect()** procedure.
2. Run one or more SQL DDL statements.
3. Disconnect from the grid by running the **ifx_grid_disconnect()** procedure.

Example

Suppose you have a retail shop with a web site. You replicate your data to several other locations for web applications. You want to be able to quickly and easily create, drop, and update tables. You create a grid named **grid1**, from which you can update the database schema for all servers in one step. The following example creates a table for special offers in the **prod_db** database:

```
Database  prod_db;

 EXECUTE PROCEDURE ifx_grid_connect('grid1');

  CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules        lvarchar);
 EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related reference:

"ifx_grid_connect() procedure" on page C-1

"ifx_grid_disconnect() procedure" on page C-4

Creating replicated tables through a grid

You can automatically create a replicate and start replication when you create a table through the grid.

If the table you are creating is a typed table, you must define a primary key.

When you enable replication while creating a table through a grid, replication is set up in the following way:

- A replicate is created for the table. The replicate name is based on the name of the source server. Use the **cdr list replicate** command to see the name.

- All servers that are members of the grid are included as participants in the replicate.
- The replicate is included in a replicate set that has the same name as the grid.
- The conflict resolution rule for the replicate is time stamp if you include the WITH CRCOLS clause. Otherwise, the conflict resolution rule is always apply.
- The ERKEY shadow columns are automatically added to the table.
- All other replicate properties are the same as the default properties of a replicate created through a template.

To set up replication:

1. Connect to the grid by running the **ifx_grid_connect()** procedure with the *ER_enable* argument set to 1.
2. Run a CREATE TABLE statement. Include the WITH CRCOLS clause if you want time stamp conflict resolution.
3. Disconnect from the grid by running the **ifx_grid_disconnect()** procedure.

The following example creates a table with replication enabled that uses the time stamp conflict resolution rule:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate   date,
    offer_enddate     date,
    offer_rules       lvarchar)
    WITH CRCOLS;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

If you need to alter or delete a database object that you created through a grid, perform those operation from within a grid context. For example, do not create a table from within a grid and then delete the table on one of the replication servers outside of a grid context. Instead, delete the table through the grid.

Related concepts:

"Conflict Resolution Rule" on page 3-6

Related tasks:

"Preparing tables without primary keys" on page 4-21

Related reference:

"ifx_grid_connect() procedure" on page C-1

"cdr define template" on page A-75

Altering replicated tables through a grid

You can alter replicated tables through a grid, whether or not the replicate was created through a grid. The replicate is automatically remastered.

If you are altering a replicate that was not created through a grid, all the replicate participants must be members of the grid.

Altering replicated tables through a grid has the following restrictions:

- You cannot drop a replicated column through a grid. To drop a replicated column, you must manually remaster the replicate and then drop the column.

- You cannot rename a replicated database. You must manually rename the database on each participant server.

If you alter a replicate that was not created through a grid, the replicate is automatically added to the grid replicate set if the replicate has the following properties:

- The replicated table schema is the same among all participants.
- The entire replicated table is replicated. Using a SELECT statement in the participant definition that does not include all the columns in the table or includes a WHERE clause is not allowed.

If the grid replicate set did not exist before this alter operation, it is created with the same name as the grid.

To alter a replicated table through a grid:

1. Connect to the grid by running the **ifx_grid_connect()** procedure with the *ER_enable* argument set to 1.
2. Run an ALTER TABLE statement.
3. Disconnect from the grid by running the **ifx_grid_disconnect()** procedure.

The following example alters the **special_offers** table to add a new column and remasters the replicate on all participants that are members of the grid:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

ALTER TABLE special_offers ADD (
   offer_exceptions varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related tasks:

"Dropping a Replicated Column" on page 8-24

"Adding an existing replicate to a grid replicate set" on page 7-7

Related reference:

"ifx_grid_connect() procedure" on page C-1

Propagating updates to data

You can change your data through a grid routine and propagate the changes to all the servers in the grid.

You can propagate updates to data on servers in the grid. By default, changes to data that are propagated through the grid are treated the same as changes to data that are made by Enterprise Replication apply threads: they are not replicated again. For example, if you propagate a DELETE statement through the grid to remove old data, you would not want the resulting deleted rows to be replicated as well. Although you can use the grid to run a DML statement, in general, use Enterprise Replication to replicate changes to replicated data.

The grid must exist and you must run the grid routines as an authorized user from an authorized server.

To propagate an SQL statement or a stored procedure that updates data, run the **ifx_grid_execute()** procedure with the DML statements or the stored procedure as the second argument.

Examples

Example 1: Reduce the price of products with low sales

In the following example, the **ifx_grid_execute()** procedure runs SQL statements that reduce the price of wool overcoats in stores that did not sell an overcoat in the last week:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1',
  'UPDATE price_table SET price = price * 0.75
   WHERE item =
   (SELECT item FROM inventory i, sales s
     WHERE i.description = "Wool Overcoat"
        AND  i.item = s.item
        AND  s.recent_sale_date <
    extend (current - Interval(7) DAY))');
```

Example 2: Purge old data

The following example purges all sales records before 2010:

```
Database retail_db;
 EXECUTE PROCEDURE ifx_grid_execute('grid1',
    'DELETE FROM sales WHERE sales_year < 2010');
```

Example 3: Run a low inventory report

The following example runs an existing stored procedure named **low_inventory()**:

```
EXECUTE PROCEDURE ifx_grid_procedure('grid1', 'low_inventory()');
```

Related reference:

"ifx_grid_execute() procedure" on page C-4

Rerunning failed grid routines

You can rerun a grid routine that failed on one or more servers in the grid.

If a grid routine failed on one or more servers in the grid, you can run the **cdr list grid** command with the **--nacks** option to see the details of why it failed. If a server in the grid is offline or is not connected to the network, then a grid routine will be pending on that server and will be run when the server is reconnected to the grid.

In some cases, you should not rerun a failed routine, because the failure is expected. For example, if a server already has the database object that a grid routine is creating, then that routine fails on that server. If a command failed on all grid servers, you can run the original command again instead of running the **ifx_grid_redo()** procedure.

The grid must exist and you must run the grid routine as an authorized user from an authorized server.

To rerun a grid routine, run the **ifx_grid_redo()** procedure.
If you run the **ifx_grid_redo()** procedure without additional arguments besides the grid name, all routines that failed are re-attempted on all the servers on which they failed. You can specify on which server to rerun routines and which routines to rerun.

Example

Suppose you have a grid, named **grid1**, that contains the servers **gserv_1** and **gserv_2**, which have a database named **db1**.

You create a dbspace named **dbsp2** on the server **gserv_1** and then create a table in that dbspace in a grid context with the following commands:

```
$ dbaccess db1 -
execute procedure ifx_grid_connect('grid1');
create table t100 (c1 int primary key) in dbsp2;
execute procedure ifx_grid_disconnect();
```

The **cdr list grid** command shows that the command failed on the server **gserv_2**:

```
$ cdr list grid grid1 --nack
Grid                    Node                    User
----------------------- ----------------------- -----------------------
grid1                   gserv_1*                user1
                        gserv_2
Details for grid grid1

Node:gserv_1 Stmtid:4 User:user1 Database:db1 2011-02-24 09:27:44
create table t100 (c1 int primary key) in dbsp2
NACK gserv_2 2011-02-24 09:27:45 SQLERR:-261 ISAMERR:-130
     Grid Apply Transaction Failure
```

The error indicates that the table could not be created because the specified dbspace does not exist.

You create a dbspace named **dbsp2** on the server **gserv_2** and run the **ifx_grid_redo()** procedure to rerun the original command on **gserv_2**:

```
$ dbaccess db1 –
execute procedure ifx_grid_redo('grid1');
```

The output of the **cdr list grid** command shows that the command succeeded on both servers:

```
$ cdr list grid grid1 -v
Grid                    Node                    User
----------------------- ----------------------- -----------------------
grid1                   gserv_1*                user1
                        gserv_2
Details for grid grid1
...
Node:gserv_1 Stmtid:4 User:user1 Database:db1 2011-02-24 09:27:44
create table t100 (c1 int primary key) in dbsp2
ACK gserv_1 2011-02-24 09:27:44
ACK gserv_2 2011-02-24 09:31:09
```

Related reference:

"ifx_grid_redo() procedure" on page C-9

"cdr list grid" on page A-94

Enabling replication within a grid transaction

You can enable replication within a transaction that is run in the context of the grid.

By default, the results of transactions run in the context of the grid are not also replicated by Enterprise Replication. In certain situations you might want to both propagate a transaction to the servers in the grid and replicate the results of the transaction.

To enable replication within a transaction:

1. Connect to the grid with the **ifx_grid_connect()** procedure.
2. Create a procedure that performs the following tasks:
   a. Defines a data variable for the Enterprise Replication state information.
   b. Runs the **ifx_get_erstate()** function and save its result in the data variable.
   c. Enables replication by running the **ifx_set_erstate()** procedure with an argument of 1.
   d. Runs the statements that you want to replicate.
   e. Resets the replication state to the previous value by running the **ifx_set_erstate()** procedure with the name of the data variable.
3. Disconnect from the grid with the **ifx_grid_disconnect()** procedure.
4. Run the newly-defined procedure by using the **ifx_grid_procedure()** procedure.

Example

Suppose that a retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory and then replicates that summary information to a central server. A stored procedure named **low_inventory()** that creates a low inventory report exists on all the servers in the grid named **grid1**. The following example creates a new procedure named **xqt_low_inventory()** that enables replication for the **low_inventory()** procedure, and then runs the **low_inventory()** procedure:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');
CREATE PROCEDURE xqt_low_inventory()
 DEFINE curstate integer;
 EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
 EXECUTE PROCEDURE ifx_set_erstate(1);
 EXECUTE PROCEDURE low_inventory();
 EXECUTE PROCEDURE ifx_set_erstate(curstate);
END PROCEDURE;
EXECUTE PROCEDURE ifx_grid_disconnect();
EXECUTE PROCEDURE ifx_grid_procedure('grid1', 'xqt_low_inventory()');
```

The following events occur in this example:

1. The **ifx_grid_connect()** procedure connects to the **grid1** grid so that the **xqt_low_inventory()** procedure is propagated to all the servers in the **grid1** grid.
2. The **xqt_low_inventory()** procedure defines a data variable called **curstate** to hold the Enterprise Replication state information.
3. The **ifx_get_erstate()** function obtains the Enterprise Replication state and stores it in the **curstate** variable. The **ifx_set_state()** procedure enables replication.
4. The **low_inventory()** procedure is run.
5. The replication state is reset back to its original value.
6. The connection to the grid is closed by the **ifx_grid_disconnect()** procedure.
7. The **ifx_grid_procedure()** procedure runs the **xqt_low_inventory()** procedure on all the servers in the grid and the result of the **low_inventory()** procedure is replicated like any normal updating activity.

Related reference:

"ifx_set_erstate() procedure" on page C-10

"ifx_get_erstate() function" on page C-1

Routing client connections in a grid

You can route client connections to servers in the grid based on the quality of replicated data and transaction latency by configuring service-level agreements for a Connection Manager.

You must install IBM Informix Client Software Development Kit on the computer you want to use for the Connection Manager.

When you create a grid, a replicate set is created with the same name as the grid. Only a grid replicate set can use the Connection Manager FAILURE and LATENCY policies based on the quality of data on each replication server.

To route client connections in a grid:

1. On a server in the grid, enable monitoring the quality of replicated data for a replication domain by running the **cdr define qod** command with the **--start** option.
2. On the Connection Manager computer, set the **INFORMIXDIR** environment variable to the directory in which the IBM Informix Client Software Development Kit is installed, and set the **INFORMIXSERVER** environment variable to any of the Enterprise Replication root servers in the grid.
3. On the Connection Manager computer, create a Connection Manager configuration file:
   a. Set the TYPE keyword to REPLSET.
   b. List the participant servers with the NODES keyword. Alternatively, you can list the participant servers in the SLA field.
   c. Create an SLA with the SLA name equal to REPLSET, a reference to the NODES list, and a policy set equal to LATENCY and FAILURE to ensure that Connection Manager selects the replication server with the lowest latency and the fewest transaction failures. The replicate set name you include in the SLA definition must be the name of a grid. You can also include a WORKLOAD policy and specify how to redirect failed client connections. The following example shows a configuration file for a replicate set named **grid1** that uses the LATENCY and FAILURE policies.

```
NAME    cm1
TYPE    REPLSET
NODES   list1=gserv1+gserv2+gserv3+gserv4
SLA     report1=REPLSET grid1 list1 <policy=LATENCY+FAILURE>
FOC     DISABLED
LOGFILE /usr/informix/tmp/cm1.log
```

4. On the Connection Manager computer, update the **sqlhosts** file to add a line about the SLA. For example, if the name of the computer running Connection Manager is **cmhost1**, and the port name is **cmport3**, the `sqlhosts` file has the following entry for the **report1** SLA:

```
report1   ontlitcp      cmhost1 cmport3
```

5. On the Connection Manager computer, update the **sqlhosts** file to add information about the server groups and the servers in the groups. If the replication server is also a primary server in a cluster, you must include information for the secondary servers in case of failover. The group and server information is the same as the `sqlhosts` entries needed on replication servers.

For example, if you have a grid with four servers and you want Connection Manager to manage connections to all of them, the `sqlhosts` file for Connection Manager might have the following entries:

```
gserv1     group    -                     -     i=143
serv1      ontlitcp ny.usa.com            1230  g=gserv1
gserv2     group    -                     -     i=144
serv2      ontlitcp tokyo.japan.com       1231  g=gserv2
gserv3     group    -                     -     i=145
serv3      ontlitcp rome.italy.com        1232  g=gserv3
gserv4     group    -                     -     i=146
serv4      ontlitcp perth.australia.com   1233  g=gserv4
```

6. On the Connection Manager computer, start a Connection Manager by running the **oncmsm** utility, specifying the configuration file. For example, the following command starts Connection Manager and uses the configuration file named `gridconfig`:

   ```
   oncmsm -c gridconfig
   ```

To maintain accurate information about the quality of data, run **cdr reset qod** before repairing inconsistent data.

Related concepts:

"Example of setting up a replication system with a grid" on page 7-2

Related reference:

"cdr reset qod" on page A-124

"cdr define qod" on page A-60

The oncmsm utility (Administrator's Reference)

Chapter 8. Managing Replication Servers and Replicates

These topics cover how to manage your Enterprise Replication system, including managing replication servers, replicates and participants, replicate sets, templates, replication server network connections, and resynchronizing data, and performing alter operations on replicated tables.

Managing Replication Servers

You manage replication servers with the **cdr** commands.

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 the **cdr list server** ***server_name*** command. For more information about the possible server states, see "cdr list server" on page A-103.

Note: When switching a server to administration mode to perform administrative tasks, be aware that any Enterprise Replication on the server will be started (or continue to run normally if already started). In this situation data on which you might be relying may change as other users modify it, and concurrency problems may arise as others access the same data. To avoid this problem, launch the server using the **oninit -Dj** command; if the server is already running, use the **cdr stop** command to shut down any currently running replications.

Related tasks:

"Setting Configuration Parameters" on page 4-14

Modifying Replication Server Attributes

To modify replication server attributes, 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 6-1. For more information about the **cdr modify server** command, see "cdr modify server" on page A-112.

Dynamically Modifying Configuration Parameters for a Replication Server

You can alter the settings for Enterprise Replication configuration parameters and environment variables on a replication server while replication is active.

Use the following commands to dynamically update values of most Enterprise Replication configuration parameters:

cdr add onconfig
: Adds a value. This option is available only for configuration parameters and environment variables that allow multiple values.

cdr change onconfig
Replaces the existing value. This option is available for all Enterprise Replication configuration parameters and environment variables.

cdr remove onconfig
Removes a specific value. This option is available only for configuration parameters and environment variables that allow multiple values.

The commands change configuration parameters in the `onconfig` file. To update environment variables, use the CDR_ENV configuration parameter.

To dynamically update the value of the CDR_DELAY_PURGE_DTC configuration parameter, use the **onmode -wf** command.

The following table shows which changes are valid for Enterprise Replication configuration parameters.

Table 8-1. Options for dynamically updating Enterprise Replication configuration parameters

Configuration Parameter	cdr add onconfig	cdr change onconfig	cdr remove onconfig
CDR_APPLY	No	No	No
CDR_DBSPACE	No	Yes	No
CDR_DSLOCKWAIT	No	Yes	No
CDR_ENV CDR_ALARMS	No	No	No
CDR_ENV CDR_LOGDELTA	No	Yes	No
CDR_ENV CDR_PERFLOG	No	Yes	No
CDR_ENV CDR_RMSCALEFACT	No	Yes	No
CDR_ENV CDR_ROUTER	No	Yes	No
CDR_ENV CDRSITES_731	Yes	Yes	Yes
CDR_ENV CDRSITES_92X	Yes	Yes	Yes
CDR_ENV CDRSITES_10X	Yes	Yes	Yes
CDR_EVALTHREADS	No	Yes	No
CDR_LOG_LAG_ACTION	Yes	Yes	Yes
CDR_LOG_STAGING_MAXSIZE	Yes	Yes	Yes
CDR_MAC_DYNAMIC_LOGS	No	Yes	No
CDR_NIFCOMPRESS	No	Yes	No
CDR_QDATA_SBSPACE	Yes	Yes	Yes
CDR_QUEUEMEM	No	Yes	No
CDR_SERIAL	No	Yes	No
CDR_SUPPRESS_ATSRISWARN	Yes	Yes	Yes
ENCRYPT_CDR	No	Yes	No
ENCRYPT_CIPHERS	No	Yes	No
ENCRYPT_MAC	Yes	Yes	Yes
ENCRYPT_MACFILE	Yes	Yes	Yes
ENCRYPT_SWITCH	No	Yes	No

You can view the setting of Enterprise Replication configuration parameters and environment variables with the **onstat -g cdr config** command.

Related reference:

"onstat -g cdr config" on page E-4

"cdr add onconfig" on page A-28

"cdr change onconfig" on page A-31

"cdr remove onconfig" on page A-121

Appendix B, "Enterprise Replication configuration parameter and environment variable reference," on page B-1

Viewing Replication Server Attributes

After you define a server for replication, 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-103.

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 the *INFORMIXSERVER* environment variable. 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 2-3 and "Connect Option" on page A-3.

Temporarily stopping replication on a server

You can temporarily stop replication on a server to perform maintenance tasks in several different ways.

You can stop Enterprise Replication on a server by shutting down the database server. Replication begins again when you restart the database server.

However, you might want to temporarily stop the Enterprise Replication threads without stopping the database server.

You can temporarily stop replication by running the **cdr stop** command. The stopped server does not capture data to be replicated. Other replication servers in the domain continue to queue replicated data for the stopped server in their send queues. Replication threads remain stopped (even if the database server is stopped and restarted) until you run the **cdr start** command. When you restart replication on the server, it receives and applies the replicated data from the other replication servers. However, if replication is stopped for long enough, the replay position on

the logical log on the stopped server can be overrun and the send queues on the active replication servers can fill up. If either of these situations happens, you must synchronize the server that was stopped.

If your replicates use time stamp or delete wins conflict resolution rules, you should temporarily stop replication on the server by using the **cdr disable server** command. Disabling a replication server is also appropriate if you do not have enough disk space to avoid overrunning the replay position. Replication servers do not queue replicated transactions for the disabled replication server, nor does the disabled replication server queue its transactions. Therefore, you must synchronize the replication server that was disabled after you enable replication on it by using the **cdr check replicateset** command. However, because information about deleted rows on the disabled replication server is saved in delete tables, you can take advantage of a time stamp repair.

Related reference:

"cdr stop" on page A-150

"cdr disable server" on page A-86

Restarting Replication on a Server

You can restart replication after Enterprise Replication was temporarily stopped.

If replication was stopped by the **cdr disable server** command, you can restart it by running the **cdr check replicateset** command with the **--repair** and the **--enable** options or by running the **cdr enable server** command. If you use the **cdr enable server** command, you must subsequently synchronize the server.

If replication stopped due to an error, you can restart replication by shutting down and restarting the database server or by running the **cdr start** command.

If replication was stopped by the **cdr stop** command, restart replication by running the **cdr start** command.

When you run the **cdr start** command, Enterprise Replication resumes evaluating the logical logs at the replay position (where Enterprise Replication stopped evaluating the logical log when the server was stopped). If the replay position was overwritten in the logical log, replication cannot restart and event alarm 75 is raised. In this situation, run the **cdr cleanstart** command to restart Enterprise Replication and then synchronize the data.

Related reference:

"cdr start" on page A-129

"cdr enable server" on page A-90

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 acknowledgment 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
```

Important: 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-158.

Resuming a Suspended Replication Server

To resume replication to a suspended server, use the **cdr resume server** command, specifying which server you want to resume. 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
```

For more information, see "cdr resume server" on page A-128.

Deleting a Replication Server

You can remove a server from an Enterprise Replication domain.

To delete a replication server on which Enterprise Replication is active, use the **cdr delete server** command. You must run the **cdr delete server** command twice: once on the server being deleted and once on another server in the domain.

To delete a replication server on which Enterprise Replication is not active, use the **cdr delete server** command with the **--force** option.

To restart Enterprise Replication on a server after you delete the server, you must define the server again with the **cdr define server** command and then synchronized the data. However, the history of rows that were deleted on the server while it was not defined for replication is lost.

Warning: Do not delete an Enterprise Replication server and immediately recreate it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the domain), failures might occur in the Enterprise Replication system at the time of the operation or later.

Examples

For example, to remove the server group **g_papeete** from Enterprise Replication, set the **INFORMIXSERVER** environment variable to server name **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. The second command connects to another server in the replication environment (**raratonga**) and deletes **g_papeete** from the **raratonga** server and all other servers in the replication domain.

Related reference:
"cdr delete server" on page A-81

Managing Replicates

You can perform the following tasks on existing replicates:

- Modify replicate attributes or participants
- View replicate properties and state
- Change the state of a replicate (whether replication is being performed)
- Delete a replicate

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 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 the **cdr change replicate --add** command. 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 the **cdr change replicate --delete** command.

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-32.

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
- Replication between servers that use different code sets

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 6-7.

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-108.

Viewing Replicate Properties

After you define a replicate, 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 detailed information on all the replicates defined on the current server. If you use the **brief** option, Enterprise Replication lists participant information about all the replicates. If you specify a replicate name, Enterprise Replication displays participant information about the replicate.

For information about this command, see “cdr list replicate” on page A-97.

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 active participants. At least two participants must be active for data replication to occur.

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:

```
 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.

When you start a replicate, you can choose to perform an initial data synchronization, as described in “Initially Synchronizing Data Among Database Servers” on page 6-20.

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.

When replication is active on an instance, you may need to double the amount of lock resources, to accommodate transactions on replicated tables.

If a replicate belongs to an exclusive replicate set, you must start the replicate set to which the replicate belongs. For more information, see “Starting a Replicate.”

For more information, see “cdr start replicate” on page A-131.

Stopping a Replicate

You can temporarily stop replication for administrative purposes.

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 transmit or process any database changes.

In general, you should only stop replication when no replication activity is likely to occur for that table or on the advice of IBM Software Support. If database activity does occur while replication is stopped for a prolonged period of time, the replay position in the logical log might be overrun. If a message that the replay position is overrun appears in the message log, you must resynchronize the data on the replication servers. For more information on resynchronizing data, see "Resynchronizing Data among Replication Servers" on page 8-14.

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.

If a replicate belongs to an exclusive replicate set, you must stop the replicate set to which the replicate belongs. For more information, see "Exclusive Replicate Sets" on page 6-18 and "Stopping a Replicate Set" on page 8-12.

Stopping a replicate set also stops any direct synchronization or consistency checking that are in progress. To complete synchronization or consistency checking, you must rerun the **cdr sync replicateset** or **cdr check replicateset** command.

For more information, see "cdr stop replicate" on page A-152.

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 in a suspended state, the replicate captures and accumulates changes to the source database, but does not transmit the captured data to the target database.

Warning: Enterprise Replication does not support referential integrity if a replicate is suspended. Instead, you should suspend a server. For more information, see "Suspending Replication for a Server" on page 8-4.

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

```
cdr suspend replicate sales_data
```

If a replicate belongs to an exclusive replicate set, you must suspend the replicate set to which the replicate belongs. For more information, see “Exclusive Replicate Sets” on page 6-18 and “Suspending a Replicate Set” on page 8-12.

For more information, see “cdr suspend replicate” on page A-155.

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
```

If a replicate belongs to an exclusive replicate set, you must resume the replicate set to which the replicate belongs. For more information, see “Exclusive Replicate Sets” on page 6-18 and “Resuming a Replicate Set” on page 8-12.

For more information, see “cdr resume replicate” on page A-126.

Deleting a Replicate

To delete the replicate, 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 the **sales_data** replicate from the global catalog, enter:

```
cdr delete replicate sales_data
```

Warning: Avoid deleting a replicate and immediately recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later. For more information, see “Asynchronous propagation considerations” on page 2-4.

Related reference:

“cdr delete replicate” on page A-79

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.

Performing an operation on a replicate set (except **cdr delete replicateset**) is equivalent to performing the operation on each replicate in the replicate set individually.

For more information, see “Managing Replicates” on page 8-6.

Routing client connections for a replicate set

You can use a Connection Manager to route client connections for the participants of a replicate set.

You must install IBM Informix Client Software Development Kit on the computer you want to use for the Connection Manager.

You can specify in your application that clients make connections through a Connection Manager instead of individual replication servers. The Connection Manager routes the client connection to an available replication server. These tasks must be performed on the Connection Manager computer.

To route client connections:

1. Set the **INFORMIXDIR** environment variable to the directory in which the IBM Informix Client Software Development Kit is installed, and set the **INFORMIXSERVER** environment variable to any of the Enterprise Replication root servers that participate in the replicate set.
2. Create a Connection Manager configuration file:
   a. Set the TYPE keyword to REPLSET.
   b. List the participant servers with the NODES keyword. Alternatively, you can list the participant servers in the SLA field.
   c. Create an SLA with the SLA name set equal to REPLSET and reference the NODES information. The following example shows a configuration file for a replicate set named **set1** that configures connection redirection.

```
NAME    cm1
TYPE    REPLSET
NODES   list1=gserv1+gserv2+gserv3+gserv4
SLA     report1=REPLSET set1 list1
FOC     DISABLED
LOGFILE /usr/informix/tmp/cm1.log
```

3. On the Connection Manager computer, update the **sqlhosts** file to add a line about the SLA. For example, if the name of the computer running Connection Manager is **cmhost1**, and the port name is **cmport3**, the `sqlhosts` file would have the following entry for the **report1** SLA:

```
report1   ontlitcp      cmhost1 cmport3
```

4. Update the **sqlhosts** file on the Connection Manager computer to add information about the server groups and the servers in the groups. If the replication server is also a primary server in a cluster, you must include information for the secondary servers in case of failover. The group and server information is the same as the `sqlhosts` entries needed on replication servers. For example, if you have a replicate set with four servers and you want Connection Manager to manage connections to all of them, the `sqlhosts` file for Connection Manager would have the following entries:

```
gserv1     group    -                     -     i=143
serv1      ontlitcp ny.usa.com            1230  g=gserv1
gserv2     group    -                     -     i=144
serv2      ontlitcp tokyo.japan.com       1231  g=gserv2
gserv3     group    -                     -     i=145
serv3      ontlitcp rome.italy.com        1232  g=gserv3
gserv4     group    -                     -     i=146
serv4      ontlitcp perth.australia.com   1233  g=gserv4
```

5. Start a Connection Manager by running the **oncmsm** utility, specifying the configuration file.

Related reference:

The oncmsm utility (Administrator's Reference)

Modifying Replicate Sets

You can modify replicate sets in two ways:

- Add or Delete Replicates
- Change Replication Frequency

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
```

The state of the replicate when you add it to a replicate set depends on the type of replicate set:

- For 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 8-9.
- For 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 or that is defined with a frequency interval, 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 "Suspending a Replicate Set" on page 8-12 and "Frequency Options" on page A-26.

For more information, see "cdr change replicateset" on page A-35.

Changing Replication Frequency For the Replicate Set

You can change the replication frequency for the replicates in an exclusive or non-exclusive replicate set using the **cdr modify replicateset** command. For more information, see "Specifying Replication Frequency" on page 6-11.

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-35.

Viewing Replicate Sets

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 8-7.

For more information, see "cdr list replicateset" on page A-101.

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:

```
set sales_set
```

When you start a replicate set, you can choose to perform an initial data synchronization, as described in "Initially Synchronizing Data Among Database Servers" on page 6-20.

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

For more information, see "cdr start replicateset" on page A-134 and "cdr start replicate" on page A-131.

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
```

Stopping a replicate set also stops any direct synchronization or consistency checking that are in progress. To complete synchronization or consistency checking, you must rerun the **cdr sync replicateset** or **cdr check replicateset** command.

For more information, see "cdr stop replicateset" on page A-154 and "cdr stop replicate" on page A-152.

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-157 and "cdr suspend replicate" on page A-155.

Related reference:

"cdr change replicateset" on page A-35

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-127 and “cdr resume replicate” on page A-126.

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 recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later. For more information, see “Asynchronous propagation considerations” on page 2-4.

Related reference:

“cdr delete replicateset” on page A-80

Managing Templates

You can use the **cdr list template** and **cdr delete template** commands to view information about your templates and to clean up obsolete templates. The commands are described in detail, including examples and sample output, in Appendix A, “The cdr Command-Line Utility Reference,” on page A-1.

You cannot update a template. To modify a template, you must delete it with the **cdr delete template** command and then re-create it with the **cdr define template** command.

Viewing Template Definitions

Use the **cdr list template** command to view detailed information about the template and the servers, databases and tables for which the template defines replication.

Deleting Templates

Use the **cdr delete template** command to delete any templates that you no longer want to use to set up replication. The command also deletes any replicate sets associated with the template which exist if the template has been realized.

Important: Deleting a template does not delete replicates that have been created by realizing a template.

Managing Replication Server Network Connections

This section explains how you can view network connections status, drop network connections, and reestablish dropped network connections.

Viewing Network Connection Status

To determine the current status of the network connection to each of the servers participating in replication, use the **cdr list server** command and look at the STATUS column of the output.

For more information, see "cdr list server" on page A-103.

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-88.

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
```

The following conditions can cause reestablishing a network connection to fail:

- A network outage
- A server is offline
- The **cdr stop**, **cdr disconnect server**, or **cdr delete server** commands were run on a server
- The system clock times on the servers differ by more than 900 seconds

For more information, see "cdr connect server" on page A-57.

Resynchronizing Data among Replication Servers

If replication has failed for some reason and data is not synchronized, there are different ways to correct data mismatches between replicated tables.

The following table compares each of the methods. All methods except manual table unloading and reloading can be performed while replication is active.

Table 8-2. Resynchronization methods

Method	Description
Direct synchronization	• Replicates all rows from the specified reference server to all specified target servers for a replicate or replicate set. • Runs as a foreground process by default, but can run as a background process. • Populates tables in a new participant. • Quickly synchronizes significantly inconsistent tables when used with the TRUNCATE statement.
Checking consistency and then repairing inconsistent rows	• Compares all rows from the specified target servers with the rows on the reference server, prepares a consistency report, and optionally repairs inconsistent rows. • Runs as a foreground process by default, but can run as a background process.
ATS or RIS file repairs	• Used to repair rows that other synchronization methods could not repair. • Repairs a single transaction at a time. • Replicates or replication server must have been configured with the ATS or RIS option.
Manual table unloading and reloading	• Manual process of unloading the target table, copying the reference table, and then loading the reference table into the target database. • Requires that replication be suspended.

Related concepts:

"Repair and Initial Data Synchronization" on page 1-3

Related reference:

"cdr stop" on page A-150

"cdr stop replicate" on page A-152

Performing Direct Synchronization

Direct synchronization replicates every row in the specified replicate or replicate set from the reference server to all the specified target servers. You can use direct synchronization to populate a new target server, or an existing target server that has become severely inconsistent.

- The Enterprise Replication network connection must be active between the Connect server, reference server and the target servers while performing direct synchronization.
- The replicate must not be in a suspended or stopped state during direct synchronization.
- The replicate must not be set up for time based replication.

You can synchronize a single replicate or a replicate set. When you synchronize a replicate set, Enterprise Replication synchronizes tables in an order that preserves

referential integrity constraints (for example, child tables are synchronized after parent tables). You can choose how to handle extra target rows and whether to enable trigger firing on target servers.

Important: Running direct synchronization can consume a large amount of space in your log files. Ensure you have sufficient space before running this command.

To perform direct synchronization, use the **cdr sync replicate** or **cdr sync replicateset** command.

You can monitor the progress of a synchronization operation with the **cdr stats sync** command if you provide a progress report task name in the **cdr sync replicate** or **cdr sync replicateset** command.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the **--name** option so that you can monitor the operation with the **cdr stats sync** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

You can significantly improve the performance of synchronizing a replicate set by synchronizing the member replicates in parallel. You specify the number of parallel processes with the **--process** option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

If direct synchronization cannot repair a row, the inconsistent row is recorded in an ATS or RIS file.

Related tasks:

"Repairing Failed Transactions with ATS and RIS Files" on page 8-21

Related reference:

"cdr sync replicate" on page A-161

"cdr sync replicateset" on page A-164

"cdr stats sync" on page A-147

Synchronizing Significantly Inconsistent Tables

If your target tables are significantly inconsistent, you can speed the synchronization process by truncating the target tables before you perform direct synchronization.

When you truncate a table by using the TRUNCATE statement, you remove all rows from the table while replication is active. After the tables on the target servers are empty, direct synchronization efficiently applies data from the source server to the target servers.

If you use the TRUNCATE statement on the supertable in a hierarchy, by default, rows in all the subtables are deleted as well. You can use the ONLY keyword to limit the truncate operation to the supertable. For more information on the TRUNCATE statement, see the *IBM Informix Guide to SQL: Syntax*.

To synchronize tables in conjunction with truncation:

1. Run the TRUNCATE statement on the tables to be synchronized on the target servers.
2. Run the **cdr sync replicate** or **cdr sync replicateset** command.

For the syntax of these commands, see "cdr sync replicate" on page A-161 and "cdr sync replicateset" on page A-164.

Checking Consistency and Repairing Inconsistent Rows

A consistency check compares the data between a reference server and one or more target servers and then generates a report that describes any inconsistencies. You can choose to repair inconsistent rows during a consistency check.

The following conditions apply when you check consistency:

- Running a consistency check can consume a large amount of space in your log files. Ensure you have sufficient space before checking consistency.
- The Enterprise Replication network connection must be active between the Connect server, reference server and the target servers while performing consistency checking and repair.
- The replicate must not be in a suspended or stopped state during consistency checking.
- The replicate must not be set up for time based replication.

You can perform a consistency check and optional synchronization on a single replicate or a replicate set. When you synchronize a replicate set, Enterprise Replication synchronizes tables in an order that preserves referential integrity constraints (for example, child tables are synchronized after parent tables). You can choose how to handle extra target rows and whether to enable trigger firing on target servers.

To perform a consistency check, use the **cdr check replicate** or **cdr check replicateset** command. Use the **--repair** option to repair the inconsistent rows. A consistency report is displayed for your review.

You can monitor the progress of a consistency check with the **cdr stats check** command if you provide a progress report task name in the **cdr check replicate** or **cdr check replicateset** command.

You can run a consistency check as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the **--name** option so that you can monitor the check with the **cdr stats check** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database. If you use the **--background** option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If synchronization during a consistency check cannot repair a row, the inconsistent row is recorded in an ATS or RIS file.

Related tasks:

"Repairing Failed Transactions with ATS and RIS Files" on page 8-21

"Indexing the ifx_replcheck Column" on page 8-19

Related reference:

"cdr check replicate" on page A-37

"cdr check replicateset" on page A-47

"cdr stats check" on page A-143

Interpreting the Consistency Report

The consistency report displays information about differences in replicated data within the replicate or replicate set.

Inconsistencies listed in the consistency report do not necessarily indicate a failure of replication. Data on different database servers is inconsistent while replicated transactions are in progress. For example, the following consistency report indicates that two rows are missing on the server **g_serv2**:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         0
g_serv2                 65         0         2         0         0

WARNING: replicate is not in sync

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

The missing rows could be in the process of being replicated from **g_serv1** to **g_serv2**.

If you choose to repair inconsistent rows during a consistency check, the report shows the condition of the replicate at the time of the check, plus the actions taken to make the replicate consistent. For example, the following report shows two missing rows on **g_serv2** and that two rows were replicated from **g_serv1** to correct this inconsistency:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         2
g_serv2                 65         0         2         0         0

Validation of repaired rows failed.
WARNING: replicate is not in sync

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

The warning indicates that inconsistencies were discovered.

The report indicates whether the replicate became consistent after the repair process. In this example, the `Validation of repaired rows failed.` message indicates that the replicate is not consistent. This might occur because some replicated transactions were still being replicated. Use the **--inprogress** option to extend the validation time.

The verbose form of the consistency report also displays the differing values for each inconsistent row.

For more information about the contents of the consistency report, see “cdr check replicate” on page A-37.

Related reference:

“cdr check replicate” on page A-37

Increase the speed of consistency checking

You can increase the speed of checking the consistency of replicates or replicate sets with the **cdr check replicate** or **cdr check replicateset** commands in several ways.

To increase the speed of consistency checking of replicate sets by checking the member replicates in parallel, use the **--process** option to set the number of parallel processes equal to the number of replicates.

To increase the speed of consistency checking by limiting the amount of data that is checked, use one or more of the following options:

- Skip the checking of large objects with the **--skipLOB** option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.
- Check from a specific time with the **--since** option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.
- When checking a replicate, you can check a subset of the data with the **--where** option.

If you have large tables, you can index the **ifx_replcheck** shadow column.

Related reference:

“cdr check replicateset” on page A-47

“cdr check replicate” on page A-37

Indexing the ifx_replcheck Column:

You can index the **ifx_replcheck** shadow column to increase the speed of consistency checking.

If you have a large replicated table, you can add the **ifx_replcheck** shadow column and then create a new unique index on that column and the existing primary key columns. The index on the **ifx_replcheck** shadow column allows the database server to determine whether rows in different tables have different values without comparing the values in those rows. You must create the index on the table in each database server that participates in the replicate.

Before you can create an index on the **ifx_replcheck** shadow column and the primary key, you must prepare the replicated table by adding the **ifx_replcheck** shadow column. You can add the **ifx_replcheck** shadow column when you create the table with the WITH REPLCHECK clause, or you can alter an existing table to add the **ifx_replcheck** shadow column with the ADD REPLCHECK clause.

You can create the index while replication is active.

To index the **ifx_replcheck** shadow column, create a unique index based on the existing primary key columns and the **ifx_replcheck** column.

For example, the following statement creates an index on a table named customer on the primary key column **id** and **ifx_replcheck**:

```
CREATE UNIQUE INDEX customer_index ON  customer(id, ifx_replcheck);
```

Related concepts:

"Preparing Tables for a Consistency Check Index" on page 4-20

Related tasks:

"Checking Consistency and Repairing Inconsistent Rows" on page 8-17

Related reference:

"cdr check replicate" on page A-37

"cdr check replicateset" on page A-47

Repair inconsistencies by time stamp

You can repair inconsistencies based on the latest time stamps among the participants instead of specifying a master server.

If your replicates use the time stamp or delete wins conflict resolution rule, you can repair inconsistencies between the participants based on the latest time stamp on any participant. If you run a time stamp repair, you do not specify a master server whose data is considered correct and to which all the other participants are matched.

To ensure that a time stamp repair is accurate, follow these guidelines:

- When you need to temporarily stop replication on a server, disable it with the **cdr disable server** command instead of stopping it with **cdr stop** command.
- If you are using the delete wins conflict resolution rule, set the CDR_DELAY_PURGE_DTC configuration parameter on all replication servers to the maximum age of modifications to rows that are being actively updated.

To run a time stamp repair, use the **cdr check replicate** or **cdr check replicateset** command with the **--repair** and **--timestamp** options. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option.

If a time stamp repair finds an extra row on any participant, the result depends on the conflict resolution rule and the last transaction for that row:

- If the conflict rule is time stamp and the most recent time stamp for the row is a delete transaction, the row will be deleted on all servers.
- If the conflict rule is time stamp and a participant has a deleted row but the most recent time stamp for that row is an update transaction, the updated row is replicated to all servers.
- If the conflict rule is delete wins and any participant has deleted that row, the row is deleted from all servers, regardless of any later update transactions.

If a time stamp repair finds mismatched rows on different servers, then the most recent update transaction for that row is replicated to the other server.

Related concepts:

"Time Stamp Conflict Resolution Rule" on page 3-7

"Delete Wins Conflict Resolution Rule" on page 3-12

Related reference:

"CDR_DELAY_PURGE_DTC configuration parameter" on page B-2

Repairing inconsistencies while enabling a replication server

If a replication server is in disabled mode, you can enable it and repair inconsistencies with the **cdr check replicateset** command.

The server must have been put in disabled mode with the **cdr disable server** command.

To enable a disabled server and synchronize it, run the **cdr check replicateset** command with the **--repair** and **--enable** options.

By default, the enable process times out after 128 seconds if the disabled replication server cannot be enabled and repaired during that time. You can specify a shorter time out period by setting the **--timeout** option to a value less than or equal to 60 seconds.

To repair all replicate sets on the disabled server, also include the **--allrepl** option and omit the **--replset** option.

Related reference:

"cdr check replicateset" on page A-47

Repairing Failed Transactions with ATS and RIS Files

You can repair failed or inconsistent transactions using an ATS or RIS file if you defined the replicate or replication server with the **–ats** or **–ris** option and the ATS or RIS files are being generated in text format.

A repair using an ATS or RIS file repairs the rows associated with the single transaction that is recorded in the specified ATS or RIS file. To apply repairs based on an ATS or RIS file, use the **cdr repair** command. The **cdr repair** command processes one ATS or RIS file each time you specify the command. The following table shows how failed operations are handled.

Failed Operation	Action Taken
Delete	Delete on the target server
Insert or Update	• If the row is found on the source server, does an update • If the row is not found on the source server, but is found on the target server, does a delete on the target server. If the row is not found on either server, performs no action.

Each operation is displayed to stderr, unless you use the **–quiet** option with the **cdr repair** command. You can preview the operations without performing them by using the **–check** option with the **cdr repair** command.

Related concepts:

"Failed Transaction (ATS and RIS) Files" on page 9-3

Appendix A, "The cdr Command-Line Utility Reference," on page A-1

"Repair and Initial Data Synchronization" on page 1-3

Related tasks:

"Performing Direct Synchronization" on page 8-15

"Checking Consistency and Repairing Inconsistent Rows" on page 8-17

Resynchronizing Data Manually

Manual resynchronization involves replacing the inconsistent table in the target database with a copy of the correct table from the reference database. Manual resynchronization is the least preferred method to repair your replicated tables because you must suspend replication to avoid producing further inconsistencies.

The following example shows how to manually resynchronize two replication database servers.

To synchronize the replication server g_papeete with the server g_raratonga

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

   See "Suspending Replication for a Server" on page 8-4.
2. Unload the table from the server group **g_raratonga.**

   See "Load and unload data" on page 4-24.
3. Load the table on **g_papeete** and specify BEGIN WORK WITHOUT REPLICATION.

   See "Load and unload data" on page 4-24 and "Blocking Replication" on page 4-17.
4. Resume replication to **g_papeete**.

   See "Resuming a Suspended Replication Server" on page 8-5.

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 4-9 and "Load and unload data" on page 4-24.

Alter, Rename, or Truncate Operations during Replication

When Enterprise Replication is active and data replication is in progress, you can perform many types of alter, rename, or truncate operations on replicated tables and databases.

Most of the supported operations do not require any special steps when performed on replicated tables or databases; some, however, do require special steps. None of the supported alter, rename, or truncate operations are replicated. You must perform these operations on each replicate participant.

You can perform the following alter, rename, and truncate operations on active, replicated tables or databases without performing extra steps:

Operation	Requirements
Add or drop default values and SQL checks	None
Add or drop fragments	Requires mastered replicate to be defined

Operation	Requirements
Add or drop unique, distinct, and foreign keys	None
Alter the locking granularity	None
Alter the next extent size	None
Change an existing fragment expression on an existing dbspace	Requires mastered replicate to be defined
Convert a fragmented table to a non-fragmented table	Requires mastered replicate to be defined
Convert a non-fragmented table to a fragmented table	Requires mastered replicate to be defined
Convert from one fragmentation strategy to another	Requires mastered replicate to be defined
Create a clustered index	Requires mastered replicate to be defined
Modify the data type of a replicated column	Requires mastered replicate to be defined
Modify the data type of a replicated column in a multiple-column primary key	Requires mastered replicate to be defined
Move a fragment expression from one dbspace to another dbspace	Requires mastered replicate to be defined
Move a non-fragmented table from one dbspace to another dbspace	Requires mastered replicate to be defined
Recluster an existing index	Requires mastered replicate to be defined
Rename a database	None
Rename a replicated column	Requires non-strict mastered replicate to be defined
Rename a table	Requires non-strict mastered replicate to be defined
Truncate a replicated table	Requires mastered replicate to be defined

You can perform the following alter operations on active, replicated tables, but you must perform extra steps, which are described in following sections:

- Add a column to a replicated table
- Drop a column from a replicated table
- Attach a fragment to a replicated table
- Change or recreate a primary key

Enterprise Replication uses shadow replicates to manage alter operations on replicated tables without causing any interruption to replication. By using shadow replicates, the replicate participants SELECT clause can be modified while replication is active. For example, a new column can be brought into the replicate definition, an existing replicated column can be removed from the replicate definition and the data type or size of a replicated column can be changed without interrupting replication. See “Defining Shadow Replicates” on page 6-9 for more information about shadow replicates.

Before altering a replicated table, ensure that you have sufficient log space allocated for long transactions, a sufficient number of locks available, and sufficient space available for the queue sbspace.

When you issue a command to alter a replicated table, Enterprise Replication places the table in *alter* mode before performing the alter operation. Alter mode is a state in which only DDL (data-definition language) and SELECT operations are allowed but DML (data-manipulation language) operations are not allowed. After the transaction that initiated the alter operation completes, Enterprise Replication unsets alter mode. Any schema changes are automatically applied to any delete tables.

The following restrictions apply when you use alter operations on replicated tables.

- Enterprise Replication must be in an active state, unless you are only adding or dropping check constraints and default values.
- Tables must have a master replicate defined.
- The DROP TABLE statement is not supported.

Recommendation: If you need to perform more than one alter operation, enclose them in a single transaction so that alter mode only needs to be set and unset one time.

For a list of common alter operation problems and how to solve them, see "Troubleshooting Tips for Alter Operations" on page 9-19.

Related reference:

"cdr alter" on page A-29

Adding a Replicated Column

You can alter a replicated table to add a new column to be replicated. The replicate must be a master replicate.

To add a new replicated column

1. Use the ALTER TABLE statement to add the column to the replicated table at all participating nodes.
2. Remaster the replicate to include the newly added column in the replicate definition, as described in "Remastering a Replicate" on page 8-27.

Dropping a Replicated Column

You can alter a replicated table to drop an existing column that is replicated. The replicate must be a master replicate.

To drop a replicated column

1. Remaster the replicate and modify the replicate's SELECT statement to remove the column being dropped, as described in "Remastering a Replicate" on page 8-27.
2. If the master replicate has shadow replicates defined, remove the column being dropped from their definitions as well.
3. Wait for the shadow replicate created by the remastering process to be cleaned up automatically. To check when the shadow replicate has been deleted, look in the server log file for a message similar to this:

   ```
   CDR CDRRTBCleaner: Deleted obsolete replicate
   Shadow_4_Repl1_GMT1090373046_GID10_PID28836
   ```

   The second line shows the name of the shadow replicate. See "cdr remaster" on page A-119 for information about the format of shadow replicate names.

Alternatively, you can use either the **cdr list replicate** or **onstat -g cat repls** command to view the status of the shadow replicate. After the shadow replicate has been deleted, these commands will no longer show information about it.

4. Use the ALTER TABLE statement to drop the columns from the replicated table at all participating nodes.

Related tasks:

"Altering replicated tables through a grid" on page 7-11

Modifying the Data Type or Size of a Replicated Column

You can modify the size or type of a replicated column for all basic data types and for the BOOLEAN and LVARCHAR extended types. Modifying the data type or size of columns of other extended types is not supported. The replicate must be a master replicate.

When you modify a replicated column, do not insert data into the modified column that will not fit into the old column definition until all participants have been altered, because the data might be truncated or data conversion to and from the master dictionary format to the local dictionary format might fail. Enterprise Replication handles the data type mismatch by having the source server convert data that is in the local dictionary format to the master dictionary format, and the target server convert data from the master dictionary format to the local dictionary format. If Enterprise Replication detects a mismatch in data type or size between the master replicate definition and the local table definition, a warning is printed in the log file.

If Enterprise Replication is not able to convert the replicated row data into the master dictionary format on the source server while queuing replicated data into the send queue, the replicate is stopped for the local participant. If this occurs, you must correct the problem and then restart the replicate from the local participant with the **--syncdatasource** option. If the correction is to delete the problematic row data, delete the row using the BEGIN WORK WITHOUT REPLICATION statement. Otherwise, the deleted row is moved from the replicated table to the associated delete table, which might cause problems for the subsequent alter operation on the replicated table.

If Enterprise Replication cannot convert row data from the master dictionary format to local table dictionary format at the target server after receiving replicated data, the replicated transaction is spooled to ATS and RIS files. For example, if you modify a SMALLINT column to an INTEGER column, make sure that you do not insert data that is too large for the SMALLINT data type until the alter operation is performed at all replicate participants, and remastering is performed so that the master dictionary reflects the INTEGER data type.

Important: While modifying a replicated column, sometimes it is possible that the alter operation on the base table succeeds, but the delete table modification might fail when Enterprise Replication unsets alter mode. If this happens, you will see a message similar to the following in the server message log file:

```
CDRGC: cannot populate data into the new delete table
SQL error=-1226, ISAM error=0
```

This situation can happen while modifying a replicated column from a data type larger in length or size to a data type smaller in length or size, for example, from an INTEGER column to a SMALLINT column, and if the delete table has data which cannot fit in the new type column.

To avoid this situation, do not convert between data types that cause data truncation or produce cases where data cannot fit into the new type. If the above situation has already occurred, carefully update or delete the problematic rows from the delete table and attempt to unset alter mode manually by using the **cdr alter** command. If you cannot resolve the problem, contact IBM Software Support.

To modify a replicated column:

1. Issue the alter command to modify the replicated column.
2. Perform the alter operation at all the replicate participants.
3. Optionally remaster the replicate to update the column definition in the replicate definition, as described in "Remastering a Replicate" on page 8-27.

After an alter operation, the master dictionary no longer matches the replicated table dictionary. Because data transfer is always done in master dictionary format, data conversion between the local dictionary format and the master dictionary format is performed. Data conversion can slow the performance of your replication system. The remastering process changes the master dictionary to match the altered replicated table dictionary. Therefore, after remastering, data conversion is not necessary.

Primary keys have special considerations. For more information, see "Considerations for Changing or Recreating Primary Key Columns."

Changing the Name of a Replicated Column, Table, or Database

You can change the name of a replicated column, table, or database while replication is active. The replicate must be a master replicate.

To change the name of a replicated column, table, or database, run the SQL statement RENAME COLUMN, RENAME TABLE, or RENAME DATABASE on all participants in the replicate.

Related reference:

RENAME TABLE statement (SQL Syntax)

RENAME COLUMN statement (SQL Syntax)

RENAME DATABASE statement (SQL Syntax)

Considerations for Changing or Recreating Primary Key Columns

There are some special considerations for changing or recreating the primary key column definition of a replicated table while replication is active, unless the replicated tables also have the ERKEY shadow columns defined.

If the table has ERKEY shadow columns, you do not need to perform any special steps to modify the primary key columns.

If the primary key contains multiple columns, you do not need to perform any special steps to modify one or more of its columns. The column modification implicitly recreates the primary key.

If the primary key is a single column, you must enclose the primary key column modification and the primary key recreation operations in a single transaction.

If you wish to drop and recreate a primary key, you must manually set alter mode, drop and recreate the primary key, and then manually unset alter mode.

Related concepts:

"Limited SQL Statements" on page 2-11

Attaching a New Fragment to a Replicated Table

To attach a new fragment, you must first manually place the replicated table in alter mode using the **cdr alter** command (described in Appendix A, "The cdr Command-Line Utility Reference," on page A-1). Enterprise Replication cannot automatically set alter mode for this operation due to an SQL restriction that requires attaching a fragment to be performed in multiple steps.

To attach a new fragment to a replicated table

1. Set alter mode on the replicate using the **cdr alter** command.
2. Drop the primary key of the table.
3. Attach the new fragment.
4. Re-create the primary key.
5. Unset alter mode using the **cdr alter** command.

Related tasks:

"Preparing tables without primary keys" on page 4-21

Remastering a Replicate

The **cdr remaster** command redefines an existing master replicate, or turns an existing non-master replicate into a master replicate. You must run the **cdr remaster** command if you add a new replicated column or drop a replicated column. If you modify a replicated column, you should remaster, however, remastering is not mandatory.

Related reference:

"cdr swap shadow" on page A-159

Automatic Remastering

To use automatic remastering run the **cdr remaster** command for the replicate for which you want to update the definition.

To use automatic remastering, the master replicate definition must have been created with name verification turned on (**--name** option of the **cdr define replicate** command set to **y**). See Appendix A, "The cdr Command-Line Utility Reference," on page A-1 for details about the **cdr remaster** command and the **cdr define replicate** command.

Manual Remastering

You must use manual remastering if your participants to not have matching column names and they were created with name verification turned off (**--name** option of the **cdr define replicate** command set to **n**).

To manually remaster a replicate

1. Use the **cdr define replicate** command to create a shadow replicate with the same attributes as the primary replicate and with the **--mirrors** option, but with

a SELECT statement that is correct for the table after the alter operation. The SELECT statement can include newly added columns or omit newly dropped columns.

2. Use the **cdr swap shadow** command to exchange the existing primary replicate and the newly created shadow replicate.

While performing the **cdr swap shadow** operation, Enterprise Replication stores the BEGIN WORK position of the last known transaction sent to the grouper as a *swap log position* for the current swap operation. Any transaction begun prior to the swap log position will use the original (old) replicate definition. Any transaction begun after the swap log position will use the new replicate definition.

The old replicate definition will be cleaned up automatically after the replicate definition is no longer required by Enterprise Replication.

Recapture replicated transactions

If you want a transaction to continue to be replicated after it reaches the target replication servers, you can use the **ifx_set_erstate()** procedure.

By default, when Enterprise Replication reads the logical logs to capture transactions, replicated transactions are ignored. For example, if a transaction is replicated from **serv1** to **serv2**, that transaction is not captured for replication on **serv2** because it has already been replicated. Replication stops when transactions reach target servers, but you can configure a transaction to be recaptured and continue to be replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Example

Suppose that a retail chain wants to run a procedure to create a report that populates a summary table of each store's current inventory and then replicates that summary information to a central server. A stored procedure named **low_inventory()** that creates a low inventory report exists on all replications servers. The following example creates a new procedure named **xqt_low_inventory()** that enables replication for the **low_inventory()** procedure, and then runs the **low_inventory()** procedure:

```
CREATE PROCEDURE xqt_low_inventory()
 DEFINE curstate integer;
 EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
 EXECUTE PROCEDURE ifx_set_erstate(1);
 EXECUTE PROCEDURE low_inventory();
 EXECUTE PROCEDURE ifx_set_erstate(curstate);
END PROCEDURE;
```

The following events occur in this procedure:

1. The **xqt_low_inventory()** procedure defines a data variable called **curstate** to hold the Enterprise Replication state information.
2. The **ifx_get_erstate()** function obtains the Enterprise Replication state and stores it in the **curstate** variable. The **ifx_set_state()** procedure enables replication.
3. The **low_inventory()** procedure is run.
4. The replication state is reset back to its original value.

When a transaction runs the **xqt_low_inventory()** procedure, the execution of the procedure is replicated to all replication servers and the result of the

low_inventory() procedure is then replicated like any normal updating activity.

Related reference:

“ifx_set_erstate() procedure” on page C-10
“ifx_get_erstate() function” on page C-1

Chapter 9. Monitoring and Troubleshooting Enterprise Replication

Enterprise Replication provides tools to help diagnose problems that arise during replications.

In This Chapter

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 E, "onstat Command Reference," on page E-1.

This chapter covers the following topics:

Related tasks:

"Setting Up Failed Transaction Logging" on page 6-11

Related reference:

"cdr define server" on page A-71

"cdr modify server" on page A-112

"cdr view" on page A-169

Monitor Enterprise Replication

You can monitor the Enterprise Replication system with several different methods, depending on your needs.

You can monitor the status of Enterprise Replication servers in the following ways:

- Use the **cdr view** command. Specify one or more subcommands, depending on what information you want to monitor.
- Use the IBM OpenAdmin Tool (OAT) for Informix with the Replication plug-in.

Use SQL queries on the system monitoring tables.

Run **onstat** commands to view local server information.

You can determine if a session thread is performing an Enterprise Replication apply or sync operation by running the **DBINFO('cdrsession')** function.

Set the ALARMPROGRAM script to capture event alarms for the following situations:

- Enterprise Replication errors
- ATS and RIS file generation

- Dropped connections between replication servers
- Replication state changes caused by Enterprise Replication commands, if state change event alarms are enabled

Related reference:

"cdr view" on page A-169

Appendix G, "SMI Tables for Enterprise Replication Reference," on page G-1

Appendix E, "onstat Command Reference," on page E-1

"Enterprise Replication Event Alarms" on page 9-21

DBINFO Function (SQL Syntax)

Solve Replication Processing Problems

Diagnose, monitor, and solve possible problems that can occur while Enterprise Replication is running.

You should understand the typical behavior of your Enterprise Replication system. There are many factors that contribute to the performance and other behaviors, including: hardware configuration, network load and speed, type of replication, and number of replicated transactions.

Use the **cdr view** command or the SMI tables to understand the typical behavior of your system, establish benchmarks, and track trends. Deviations from typical behavior do not necessarily indicate a problem. For example, transactions might take longer to replicate during peak usage times or during end-of-month processing.

The following table describes some replication processing problems that might occur.

Table 9-1. Potential Replication Problems and Solutions

Problem	How to diagnose	How to solve
Enterprise Replication is not running	• Run the **cdr view state** command • Query the **syscdr_state** SMI table • Examine event alarms captured by the alarm program	Start replication with the **cdr start** command.
One or more Enterprise Replication servers are not running or connected to the network	• Run the **cdr view servers** command • Run the **cdr view nif** command • Query the **syscdr_nif** SMI table • Examine event alarms captured by the alarm program	Start the database server or fix the connection problem.

Table 9-1. Potential Replication Problems and Solutions (continued)

Problem	How to diagnose	How to solve
Replicated transactions failed	Determine if there are ATS or RIS files: • Look at the ATS and RIS directories on the local server for the existence of ATS or RIS files • Run the **cdr view atsdir risdir** command to see the number of ATS and RIS files for each server • Query the **syscdr_atsdir** or **syscdr_risdir** SMI table for a specific server • Examine event alarms captured by the alarm program	Run the **cdr repair** command. See "cdr repair" on page A-122.
Transactions are spooling to disk	Determine how much spool memory is being used: • Run the **cdr view profile** command to see the status of all queues on all servers • Run the **cdr view sendq** command to see the status of the send queue on all servers • Run the **cdr view rcv** command to see the status of the receive queue on all servers	See "Increasing the Sizes or Numbers of Storage Spaces" on page 9-17.
Potential log wrap situation	Determine how many log pages must be used before Enterprise Replication reacts a potential log wrap situation: • Run the **cdr view ddr** command to see the number of unused log pages for all servers • Query the **syscdr_ddr** SMI table to see the number of unused log pages for a specific server	See "Handle potential log wrapping" on page 9-15.

If you do need to call IBM Software Support, find the version of the database server that is running Enterprise Replication with the **cdr -V** command.

Failed Transaction (ATS and RIS) Files

Aborted Transaction Spooling (ATS) and Row Information Spooling (RIS) files can be generated when replicated transactions fail.

You can use the ATS and RIS files to identify problems or as input to the **cdr repair** command or custom utilities that extract or reapply the aborted rows.

When ATS or RIS file generation is enabled for a replicate, all failed replication transactions are recorded in ATS or RIS files. Each ATS file contains all the information pertinent to a single failed transaction, while each RIS file contains information about a single failed row. If a replicated transaction fails for any reason (constraint violation, duplication, and so forth), all the buffers in the replication message that compose the transaction are written to a local file.

ATS file generation occurs if the entire transaction is aborted. Transactions defined with row scope that have aborted rows but are successfully 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, if RIS is enabled.

RIS files can contain the following types of information:

- 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 forth)
- Special SPL routine return codes, as defined by the application (if an SPL routine is called to resolve a conflict)

In some cases, such as with long transactions, the database server itself aborts transactions. In these cases, Enterprise Replication does *not* generate an ATS or RIS file.

ATS and RIS files can be generated under the following circumstances:

- ATS or RIS generation is enabled for a replicate, the replicate uses a conflict resolution rule other than ignore or always-apply, and a conflict is detected on a target server.
- Under some error conditions, ATS or RIS files can be generated on a source server, regardless if ATS or RIS generation is enabled or the conflict resolution rule.

When an ATS or RIS file is generated, an event alarm with a class ID for 48 is also generated. You can use event alarms to send notifications to a database administrator.

Related concepts:

"Conflict Resolution Scope" on page 3-14

Related tasks:

"Repairing Failed Transactions with ATS and RIS Files" on page 8-21

Related reference:

"CDR_DISABLE_SPOOL Environment Variable" on page B-16

"cdr define replicate" on page A-61

Enabling ATS and RIS File Generation

You can enable the generation of ATS and RIS files when you define a replicate.

Failed transactions are not automatically recorded in ATS and RIS files. You can choose to generate either ATS or RIS files, or both.

You should create a separate directory to store ATS and RIS files. If you do not create a separate directory and specify it when you define the replication server, Enterprise Replication stores the ATS and RIS files in the **/tmp** directory on UNIX and the **%INFORMIXDIR%\tmp** directory on Windows.

To collect ATS and RIS information

1. Create a directory for Enterprise Replication to store ATS and RIS files. You can create two directories if you want to generate both types of file and store them in separate directories.
   - If you are using primary-target replication, create the directory on the target system.
   - If you are using update-anywhere replication and have a conflict resolution rule other than ignore or always-apply enabled, create the directory on all participating replication systems.
2. When you define or modify a replication server, specify the location of the ATS and RIS directory by using the **--ats** and **--ris** options of the **cdr define server** command or the **cdr modify server** command.
3. When you define or modify a replicate, specify that ATS and RIS file generation is enabled by using the **--ats** and **--ris** options of the **cdr define replicate** command or the **cdr modify replicate** command.

Related tasks:

"Creating ATS and RIS Directories" on page 4-13

Related reference:

"cdr define server" on page A-71

"cdr define replicate" on page A-61

"cdr modify server" on page A-112

"cdr modify replicate" on page A-108

ATS and RIS File Names

Each ATS and RIS file has a unique name based on the conditions under which it was generated.

The following table provides the naming convention for ATS and RIS files:
`type.target.source.threadID.timestamp.sequence.extension`

Table 9-2. ATS and RIS file naming conventions

Name	Description
type	The format of the file: `ats` or `ris`.
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 or RIS file was generated.
sequence	A unique integer, incremented each time an ATS or RIS file is generated.
extension	The file type. No extension indicates a text file; `xml` indicates an XML file.

The naming convention ensures that all ATS and RIS file names that are generated are unique. However, when an ATS or RIS file is opened for writing, any previous file contents are overwritten. (Enterprise Replication does not append to a spool file; if a name collision does occur with an existing file, the original contents of the file are lost.)

The default delimiter for the *timestamp* portion of text file names is a colon (`:`) on UNIX and a period (`.`) on Windows. You can define the delimiter between the

hour, minute, and second values with the CDR_ATSRISNAME_DELIM environment variable. XML files always use a period (.) delimiter between the hour, minute, and second values.

The following is an example of a name of an ATS file in text format on UNIX for a transaction sent by server **g_amsterdam** to server **g_beijing**:

```
ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6
```

The following is an example of the same ATS file name in XML format:

```
ats.g_beijing.g_amsterdam.D_2.000529_23.27.16.6.xml
```

The following is an example of a similar RIS file name in XML format:

```
ris.g_beijing.g_amsterdam.D_2.000529_23.27.16.5.xml
```

Related reference:

"CDR_ATSRISNAME_DELIM Environment Variable" on page B-16

ATS and RIS File Formats

You can choose to generate ATS and RIS files in text format, XML format, or both formats.

The format of ATS and RIS files is part of the server definition that you create with the **cdr define server** command:

Text (Default)
ATS and RIS files are generated as text files that Enterprise Replication can process during a repair operation. Text format is useful if you intend to use the **cdr repair** command to repair inconsistencies.

XML ATS and RIS files are generated as XML files that you can use if you write your own custom repair scripts. You cannot use ATS or RIS files in XML format with the **cdr repair** command.

Both ATS and RIS files are generated in both text and XML format so that you can choose how to process failed transactions.

Enterprise Replication raises event alarms when ATS and RIS files are generated regardless of format.

XML File Format

The information in ATS and RIS files that are in XML format is organized in specific XML tags.

The XML format uses an XML schema that is stored in the **INFORMIXDIR/etc** directory.

Data in XML files uses the UTF-8 encoding format.

Columns that appear empty could contain a null value or an empty string. The XML format differentiates between null data and empty strings by setting the isNull="true" attribute of the COLUMN tag for null data.

Data Types That are Not Shown

The values of the following data types are not shown in XML files:

- Smart large objects
- Simple large objects

- User-defined data types

For these data types, the following attributes are set for the COLUMN tag:

- isLOBorUDT="true"
- dataExists="false"

Special Symbols

The following symbols are replaced if they exist in row data:

- < is replaced by <
- > is replaced by >
- & is replaced by &
- " is replaced by "
- ' is replaced by '

Example

The following example shows an ATS file displaying a transaction with two failed insert operations. The third column in each row contains a data type that is not shown.

```
<?xml version="1.0" encoding="UTF-8"?>
<ERFILE version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/informix/etc/idser.xsd">
  <ATS version="1">
    <TRANSACTION RISFile="/tmp/ris.g_cdr_ol_3.g_cdr_ol_2.D_5.080411_14.08.57.3.xml"
                 generateRISFile="true" processedRows="2">
       <SOURCE id="20" name="g_cdr_ol_2" commitTime="2008-04-11T14:08:57"/>
       <TARGET id="30" name="g_cdr_ol_3" receiveTime="2008-04-11T14:08:57"/>
       <MESSAGE>All rows in a transaction defined with row scope were rejected</MESSAGE>
     </TRANSACTION>
     <ATSROWS>
       <ATSROW num="1" replicateID="655362" database="bank" owner="testadm" table="customer"
              operation="Insert">
         <REPLICATED>
           <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                          cdrTimeString="2008-04-11T14:08:57"/>
           <DATA>
             <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">261</COLUMN>
             <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">cdr_ol_2</COLUMN>
             <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                     isNull="false"></COLUMN>
           </DATA>
         </REPLICATED>
       </ATSROW>
       <ATSROW num="2" replicateID="655362" database="bank" owner="testadm" table="customer"
              operation="Insert">
         <REPLICATED>
           <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                          cdrTimeString="2008-04-11T14:08:57"/>
           <DATA>
             <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">262</COLUMN>
             <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                     isNull="false">cdr_ol_2</COLUMN>
             <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                     isNull="false"></COLUMN>
           </DATA>
         </REPLICATED>
```

```
      </ATSROW>
    </ATSROWS>
  </ATS>
</ERFILE>
```

The following example shows the corresponding RIS file for the failed transaction shown in the ATS example.

```
<?xml version="1.0" encoding="UTF-8"?>
<ERFILE version="1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="/usr/informix/etc/idser.xsd">
  <RIS version="1">
    <SOURCE id="20" name="g_cdr_ol_2" commitTime="2008-04-11T14:08:57"/>
    <TARGET id="30" name="g_cdr_ol_3" receiveTime="2008-04-11T14:08:57"/>
    <RISROWS>
      <RISROW num="1" replicateID="655362" database="bank" owner="testadm" table="customer"
            operation="Insert">
        <CDRERROR num="0"/>
        <SQLERROR num="-668"/>
        <ISAMERROR num="-1"/>
        <SPLCODE num="63"/>
        <LOCAL>
          <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1206852121"
                         cdrTimeString="2008-04-11T12:08:57"/>
          <DATA>
            <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">261</COLUMN>
            <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">cdr_ol_2</COLUMN>
            <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                    isNull="false"></COLUMN>
          </DATA>
        </LOCAL>
        <REPLICATED>
          <SHADOWCOLUMNS serverID="20" serverName="g_cdr_ol_2" cdrTimeInt="1207940937"
                         cdrTimeString="2008-04-11T14:08:57"/>
          <DATA>
            <COLUMN name="col1" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">261</COLUMN>
            <COLUMN name="col2" dataExists="true" isHex="false" isLOBorUDT="false"
                    isNull="false">cdr_ol_2</COLUMN>
            <COLUMN name="col3" dataExists="false" isHex="false" isLOBorUDT="true"
                    isNull="false"></COLUMN>
          </DATA>
        </REPLICATED>
      </RISROW>
    </RISROWS>
    <TXNABORTED ATSFile="/tmp/ats.g_cdr_ol_3.g_cdr_ol_2.D_5.080411_14.08.57.4.xml"
                generateATSFile="true"/>
  </RIS>
</ERFILE>
```

XML Tags:

XML tags are used in ATS and RIS files that are generated in XML format.

Table 9-3. XML tags in ATS and RIS files

Tag name	Description	Attributes	Parent tag	Child tags
ERFILE	Top level tag for ATS and RIS files	`version`: XML file format version number.	None	ATS RIS
ATS	Parent tag for ATS files	`version`: ATS file format version number.	ERFILE	TRANSACTION ATSROWS

Table 9-3. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
RIS	Parent tag for RIS files	• `version`: RIS file format version number. • `fromSource`: Set to `true` if the RIS file is generated at the source server.	ERFILE	SOURCE TARGET RISROWS TXNABORTED TXNCOMMITTED
TRANSACTION	Contains the name of the RIS file (if it exists) and the number of rows processed before the transaction was aborted.	• `RISFile`: The name of the RIS file, if it was created. • `generateRISFile`: Set to `true` if an RIS file exists for this aborted transaction. • `processedRows`: Number of rows processed before the transaction was aborted.	ATS	SOURCE TARGET MESSAGE CDRERROR SQLERROR ISAMERROR SPLCODE
ATSROWS	Contains the replicated aborted rows	None	ATS	ATSROW
SOURCE	Contains source server information	• `id`: Server ID. • `name`: Server group name. • `commitTime`: Transaction commit time.	TRANSACTION RIS	None
TARGET	Contains target server information	• `id`: Server ID. • `name`: Server group name. • `receiveTime`: Transaction receive time.	TRANSACTION RIS	None
SQLERROR	Contains the SQL error code	`num`: Error number.	TRANSACTION RISROW	None
ISAMERROR	Contains the ISAM error code	`num`: Error number.	TRANSACTION RISROW	None
CDRERROR	Contains the data sync error code	• `num`: Error number. • `description`: Error description.	TRANSACTION RISROW	None
MESSAGE	Contains the notification message	None	TRANSACTION RISROW	None
SPLCODE	Contains the SPL code number if a stored procedure conflict rule is being used	`num`: SPL code number.	TRANSACTION RISROW	None
RISROWS	Contains the local and replicated aborted rows	None	RIS	RISROW

Table 9-3. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
RISROW	Contains information about local or replicated row data for one aborted row	• num: Row sequence number. • replicateID: Replicate ID. • database: Database name. • owner: Table owner name. • table: Table name. • operation: DML operation type.	RISROWS	MESSAGE CDRERROR SQLERROR ISAMERROR SPLCODE MESSAGE LOCAL REPLICATED
LOCAL	Contains the local row data for an aborted row	None	RISROW	SHADOWCOLUMNS DATA
REPLICATED	Contains replicated row data for an aborted row	None	ATSROW RISROW	SHADOWCOLUMNS DATA
ATSROW	Contains information for one replicated aborted row	• num: Row sequence number. • replicateID: Replicate ID. • database: Database name. • owner: Table owner name. • table: Table name. • operation: DML operation type.	ATSROWS	REPLICATED
SHADOWCOLUMNS	Optional shadow column values for local and replicated rows	• serverID: Server ID. • serverName: Server group name. • cdrTimeInt: The **cdrtime** column value in integer format (GMT time). • cdrTimeString: Time in string format. For example: 2008-11-08T20:16:25.	LOCAL REPLICATED	None
DATA	Contains aborted row data	dataExists: Identifies whether data exists for this row or not.	ATSROW RISROW	COLUMN

Table 9-3. XML tags in ATS and RIS files (continued)

Tag name	Description	Attributes	Parent tag	Child tags
COLUMN	Contains column data for an aborted row	• `name`: The column name. • `dataExists`: Identifies whether data is displayed for this column or not. • `isLOBorUDT`: Set to `true` if the column is of type UDT, smart large object or simple large object. If set to `true`, data for the column is skipped and the `dataExists` value is set to `false`. • `isHex`: Set to `true` if column data is displayed in hex format because Enterprise Replication does not have enough information to interpret the row data. • `isNull`: Set to `true` if the column value is NULL. Set to `false` if the column has a valid value or an empty string.	DATA	None
TXNABORTED	Indicates that the replicated transaction was aborted	• `ATSFile`: The name of the ATS file if the transaction was aborted and an ATS file was created for this aborted row. • `generateATSFile`: Set to `true` if an ATS file was created. • `TxnErr`: Error description for the aborted transaction.	RIS	None
TXNCOMMITTED	Indicates that the replicated transaction was committed	`totalRows`: Total number of rows processed.	RIS	None

ATS and RIS Text File Contents

The information in ATS and RIS text files is listed in rows prefaced by information labels.

The first three characters in each line of the ATS and RIS file describe the type of information for the line, as the following table defines. The first four labels apply to both ATS and RIS files. The last three labels only apply to RIS files.

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.

Label	Name	Description
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 was 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 **cdr 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 hexadecimal format if a row includes replicated UDT columns.
LRH	Local-row header	RIS only. Indicates if the local row is found in the delete table and not in the target table
LRS	Local-row shadow columns	RIS only. Contains the server ID and the time when the row was updated on the target server This line is printed only if the replicate is defined with a conflict resolution rule.
LRD	Local-row data	RIS only. 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.

Changed Column Information

If you define a replicate to only replicate columns that changed, the RRD entry in the ATS and RIS file shows 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 6-12.

BLOB and CLOB Information

If a replicate includes one or more BLOB or CLOB columns, the RRD entry in the ATS and RIS file displays the smart large object metadata (the in-row descriptor of the data), not the smart large object itself, in hexadecimal format.

BYTE and TEXT Information

When the information recorded in the ATS or RIS 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 tblspace, 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 tblspace.

```
<500 (NoChange), TEXT, PB 877(necromsv) 840338478(00/08/17 20:21:18)>
```

Example 2

In this example, 500 (NoChange)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.

UDT Information

If a replicate includes one or more UDT columns, the RRD entry in the ATS and RIS files displays the row data in delimited format as usual, except the string <skipped> is put in place of UDT column values. For example, for a table with columns of type INTEGER, UDT, CHAR(10), UDT, the row might look like this:

```
RRD 334|<skipped>|amsterdam|<skipped>
```

Disabling ATS and RIS File Generation

You can prevent the generation of ATS or RIS files, or both.

To prevent the generation of both ATS and RIS files, set the CDR_DISABLE_SPOOL environment variable to 1.

To prevent the generation of either ATS or RIS files, set the ATS or RIS directory to **/dev/null** (UNIX) or **NUL** (Windows) with the **cdr define server** or **cdr modify server** commands.

Related reference:

"cdr modify server" on page A-112

"cdr define server" on page A-71

"CDR_DISABLE_SPOOL Environment Variable" on page B-16

Suppressing Data Sync Errors and Warnings

You prevent certain data sync errors and warnings from appearing in ATS and RIS files by using the CDR_SUPPRESS_ATSRISWARN configuration parameter.

For more information on the CDR_SUPPRESS_ATSRISWARN configuration parameter, see "CDR_SUPPRESS_ATSRISWARN Configuration Parameter" on page B-14.

For a list of error and warning messages that you can suppress, see Appendix J, "Data Sync Warning and Error Messages," on page J-1.

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*, *replicate information*, and *row data*. Spooled transaction records and replicate information are stored in the transaction tables and the replicate information tables in a single dbspace. Spooled row data is stored in one or more sbspaces.

For more information, see "Setting Up Send and Receive Queue Spool Areas" on page 4-10.

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 8-3 and "cdr list server" on page A-103.
- 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 8-7 and "cdr list replicate" on page A-97.
- Enterprise Replication is replicating large transactions.

  Enterprise Replication is optimized to handle small transactions efficiently. Very large transactions or batch jobs force Enterprise Replication into an exceptional processing path that results in spooling. For best results, avoid replicating these types of transactions.

  For more information, see "Large Transactions" on page 2-11.

- 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 *IBM Informix Administrator's Guide*. For more information on configuring your logical log files for use with Enterprise Replication, see "Logical Log Configuration Guidelines" on page 4-8.
- 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.

Handle potential log wrapping

The potential for log wrap occurs when Enterprise Replication log processing lags behind the current log and the Enterprise Replication replay position is in danger of being overrun.

There are two log positions you should be aware of: the snoopy log position, which is the log position that keeps track of transactions being captured for replication, and the log replay position, which is the log position that keeps track of which transactions have been applied.

A potential log wrap situation is usually caused by the logical logs being misconfigured for the current transaction activity or by the Enterprise Replication system having to spool more than usual. More-than-usual spooling could be caused by one of the following situations:

- A one-time job might be larger than normal and thus require more log space.
- One of the target servers is currently unavailable and more spooling of replicated transactions is required.
- The spool file or paging space could be full and needs to be expanded.

You can configure how Enterprise Replication responds to a potential log wrap situation by specifying one or more of the following solutions, in order of priority, with the CDR_LOG_LAG_ACTION configuration parameter:

- Block user transactions until Enterprise Replication log processing advances far enough that the danger of log wrapping is diminished. Blocking user transactions prevents the current log position from advancing. This solution increases user response time. When user transactions are blocked, event alarm 30 unique ID 30002 is raised and the following message appears in the online log:

  ```
  DDR Log Snooping - DDRBLOCK phase started, userthreads blocked
  ```
- Compress the logical logs and save them to a log staging directory. Log files in the staging directory are deleted after they are no longer required by Enterprise Replication. You must specify the location and maximum size of the log staging directory. This solution uses very little additional disk space to temporarily save log files until the danger of log wrapping is over. The staged log files are deleted after advancing the log replay position.

  If log staging is configured, Enterprise Replication monitors the log lag state and stages log files even when Enterprise Replication is inactive.
- Dynamically add logical logs. This solution requires enough free space to be available in the logical log dbspace to add dynamic logs. You can specify how

many dynamic logical logs to add. You must manually drop the dynamic log files when the danger for log wrapping is over.

- Ignore the potential for log wrap. This solution shuts down Enterprise Replication when an overrun of the snoopy log replay position is detected. Enterprise Replication continues to function if the log replay position is overrun. If the snoopy replay position is overrun, Enterprise Replication is stopped, event alarm 47 is raised, and the following message appears in the message log file:

  ```
  WARNING:  The replay position was overrun, data may not be replicated.
  ```

  If the replay position is overrun, restart Enterprise Replication with the **cdr cleanstart** command to reset replay position to current log position and synchronize the data.
- Stop Enterprise Replication on the affected server as soon as it is detected that the log replay position is running behind. When you are ready to restart Enterprise Replication it is necessary to run the **cdr cleanstart** command only if the log replay position was overrun.

For example, you can specify that during a potential log wrap situation, Enterprise Replication stages compressed logical logs. If the log staging directory reaches its maximum size, then logical logs are added. If the maximum number of logical logs are added, then Enterprise Replication blocks user transactions. Not all options can be combined together in every possible priority order. For example, specifying to stop Enterprise Replication, to ignore the potential for log wrap, or to block user actions must always be either the only option or the last option in the list.

Related concepts:

"Logical Log Configuration Guidelines" on page 4-8

Related reference:

"CDR_LOG_LAG_ACTION Configuration Parameter" on page B-5

"CDR_LOG_STAGING_MAXSIZE Configuration Parameter" on page B-9

"CDR_MAX_DYNAMIC_LOGS Configuration Parameter" on page B-10

Monitoring Disk Usage for Send and Receive Queue Spool

Periodically monitor disk usage for the dbspace.

The sbspace that Enterprise Replication uses to spool the queues to disk is specified by the CDR_QDATA_SBSPACE configuration parameter.

To check disk usage for the spooling sbspace, run one or more of the following commands:

- **onstat -g rqm SBSPACES**
- **onstat -d**

  Tip: When you use the **onstat -d** command 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.
- The **oncheck** command with the **-cs**,**-cS**, **-ce**, **-pe**, **-ps**, and **-pS** options

Related concepts:

Manage sbspaces (Administrator's Guide)

The oncheck Utility (Administrator's Reference)

Related reference:

"onstat -g rqm" on page E-17

onstat -d command: Print chunk information (Administrator's Reference)

"CDR_QDATA_SBSPACE Configuration Parameter" on page B-12

Increasing the Sizes or Numbers 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. You can also add additional sbspaces for Enterprise Replication.

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.

See the sections on adding chunks to dbspaces and sbspaces in the *IBM Informix Administrator's Guide* and the *IBM Informix Administrator's Reference* for more information.

To increase the number of sbspaces that can be used for Enterprise Replication, create new sbspaces with the **onspaces -c -S** command and then add their names to the CDR_QDATA_SBSPACE configuration parameter with the **cdr add onconfig** command. For more information, see "cdr add onconfig" on page A-28.

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 9-14) or you must allocate additional disk space ("Increasing the Sizes or Numbers of Storage Spaces") before you can continue replication.

Common Configuration Problems

If you experience problems setting up Enterprise Replication, check the configuration of your environment and database.

To solve configuration problems:

- 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 4-10 and "CDR_QDATA_SBSPACE Configuration Parameter" on page B-12.
- Verify that the trusted environment is set up correctly.

For more information, see "Configuring secure connections for replication servers" on page 4-4.

- 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 "Database Server Groups" on page 4-2.

- 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 domain.

  For more information, see "Database Server Groups" on page 4-2.

- 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 4-16.

- 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** 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 2-5.

- For replicates that use any conflict-resolution rule except ignore and always-apply, 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 4-19.

- 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 "Participant definitions" on page 6-7 and "Participant and participant modifier" on page A-4.

- 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 2-13.

- Verify that all tables defined in a replicate have one PRIMARY KEY.

  For more information, see "Primary Key Constraint" on page 2-7, the *IBM Informix Database Design and Implementation Guide*, and *IBM Informix Guide to SQL: Syntax*.

- If high-availability clusters are also in use in the domain, then all row data sbspaces must be created with logging by using the **-Df "LOGGING=ON"** option of the **onspaces** command.

  For more information, see "Row Data sbspaces" on page 4-10 and the *IBM Informix Administrator's Guide*.

Troubleshooting Tips for Alter Operations

Alter operations on replicated tables might result in errors.

The following problems illustrate common issues with performing alter operations on replicated tables:

- **Problem:** You receive an error that the replicate is not defined after running the following command:

  ```
  cdr alter -o test:tab
  Error:Replicate(s) not defined on table test:.tab
  ```

  The owner name is missing from the table name, **test:tab**.

  Solution: Include the table owner name, for example:

  ```
  cdr alter -o test:user1.tab
  ```

- **Problem:** You receive an error that the replicated table is in alter mode after running the following command:

  ```
  > insert into tab values(1,1);

  19992: Cannot perform insert/delete/update operations on a replicated table
  while the table is in alter mode
  Error in line 1 Near character position 27
  >
  ```

  The table (**tab**) is in alter mode. DML operations cannot be performed while the table is in alter mode.

  Solution: Wait for the table to be altered and then issue the DML operation. If no alter statement is in progress against the table, then unset alter mode on the table using the **cdr alter --off** command. For example:

  ```
  cdr alter --off test:user1.tab
  ```

  You can check the alter mode status using the **oncheck -pt** command. For example:

  ```
  $ oncheck -pt db1:user1.t1

  TBLspace Report for db1:user1.t1

      Physical Address             1:63392
      Creation date                02/01/2011 16:02:00
      TBLspace Flags               400809     Page Locking
                                              TBLspace flagged for replication
                                              TBLspace flagged for CDR alter mode
                                              TBLspace use 4 bit bit-maps
      Maximum row size              4
  ...
  ```

- **Problem:** How can you tell if a replicate is a mastered replicate?

  Solution: You can check the alter mode status using the **oncheck -pt** command. For example:

  ```
  oncheck -pt test:nagaraju.tab
  ```

- **Problem:** How can you tell if a replicate is a mastered replicate?

  Solution: When you execute the **cdr list repl** command, it shows that the REPLTYPE is Master for master replicates. For example:

```
$cdr list repl
CURRENTLY DEFINED REPLICATES
------------------------------
REPLICATE: rep2
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab12
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: rep1
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
```

In the above output, **rep1** is defined as a non-master replicate and **rep2** is defined as master replicate.

- **Problem:** An alter operation on a replicated table fails.

  For example:

```
$dbaccess test -

Database selected.

> alter table tab add col4 int;

19995: Enterprise Replication error encountered while setting alter mode. See
message log file to get the Enterprise Replication error code
Error in line 1Near character position 27
>
```

  The message log output is:

```
12:36:09 CDRGC: Classic replicate rep1 found on the table test:nagaraju.tab
12:36:09 CDRGC:Set alter mode for replicate rep1
12:36:09 GC operation alter mode set operation on a replicated table failed:
Classic replicate(s) (no mastered dictionary) found on the table.
```

  Solution: The above message shows that there is a classic replicate, **rep1**, defined on the table (**tab**). Adding a new column to a replicated table is allowed when only master replicates are defined for the table.

  To perform the above alter operation, first convert the classic replicate to a master replicate. You can convert the replicate definition of **rep1** to a master replicate by issuing the following command:

```
cdr remaster -M g_delhi rep1 "select * from tab"
```

  Now look at the **cdr list repl** output:

```
$cdr list repl
CURRENTLY DEFINED REPLICATES
------------------------------
REPLICATE: rep1
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: rep2
```

```
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab12
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Master

REPLICATE: Shadow_4_rep1_GMT1112381058_GID100_PID29935
STATE: Active ON:delhi
CONFLICT: Timestamp
FREQUENCY: immediate
QUEUE SIZE: 0
PARTICIPANT: test:nagaraju.tab
OPTIONS: transaction,ris,ats,fullrow
REPLTYPE: Shadow
PARENT REPLICATE: rep1
```

You can see that **repl1** has been converted to a master replicate. You can also see that a new replicate definition, **Shadow_4_rep1_GMT1112381058_GID100_PID29935**, was also created against the table (**tab1**). Notice the last two fields of the output for **Shadow_4_rep1_GMT1112381058_GID100_PID29935**:

```
REPLTYPE: Shadow
PARENT REPLICATE: rep1
```

The Shadow attribute indicates that this replicate is a shadow replicate, and PARENT REPLICATE: **rep1** shows that this is a shadow replicate for the primary replicate **rep1**. Notice that the Master attribute is not present for this replicate definition. This shadow replicate is actually the old non-master replicate. The **cdr remaster** command created a new master replicate, **rep1**, for the table **tab** and converted the old non-master replicate (**rep1**) to a shadow replicate for the new master replicate.

This table is not yet ready to be altered because there is still a non-master replicate, **Shadow_4_rep1_GMT1112381058_GID100_PID29935**, defined for the table, **tab**. You must wait for **Shadow_4_rep1_GMT1112381058_GID100_PID29935** to be deleted automatically by Enterprise Replication after all the data queued for this shadow replicate is applied at all the replicate participants. This process can take some time. Alternatively, if you are sure that there is no data pending for this old non-master replicate, then you can issue the **cdr delete repl** command against **Shadow_4_rep1_GMT1112381058_GID100_PID29935**.

After making sure that **Shadow_4_rep1_GMT1112381058_GID100_PID29935** no longer exists, you can attempt the **ALTER TABLE tab add col4 int;** statement against the table.

Enterprise Replication Event Alarms

Certain Enterprise Replication errors and other actions generate event alarms. You can use event alarms specific to Enterprise Replication to automate many administrative tasks.

You can set your alarm program script to capture Enterprise Replication class IDs and messages and initiate corrective actions or notifications for each event. For example, you can add a chunk to the queue data sbspace or dbspace if you detect (using class ID 31) that the storage space is full.

Most event alarms operate in the background. For events that operate in the foreground, the session that triggered the alarm is suspended until the alarm program execution completes. For information on setting alarm program scripts to capture events, see "Event alarms" in the *IBM Informix Administrator's Reference*.

Many Enterprise Replication event alarms are enabled by default, but most state change event alarms are disabled by default. You can control which Enterprise Replication event alarms are enabled with the CDR_ALARMS environments variable.

The following table lists the information about Enterprise Replication event alarms:

- The class ID is an integer value identifying the category of the event.
- The event ID is a unique identifier for the specific message.
- The class message provides general information about the event.
- The specific message provides detailed information about the event.
- The severity describes the seriousness of the event on a scale from 1 to 5, where 5 is the most serious.
- Whether the event operates in the foreground and explanations for the events.
- Whether the event is disabled by default.

Table 9-4. Enterprise Replication Event Alarms

Class ID and Event ID	**Class and Specific Messages**	**Severity**	**Explanation**
Class ID: 30 **Event ID:** 30002	**Class message:** DDR subsystem notification **Specific message:** DDR Log Snooping - Catchup phase started, userthreads blocked	3	User transactions are being blocked to prevent the database server from overwriting a logical log that Enterprise Replication has not yet processed. **Online log:** The following message appears in the online log: `DDR Log Snooping - DDRBLOCK phase started, userthreads blocked` **ER state:** Active and replicating data. User transactions are temporarily blocked. **User action:** None. For information about preventing this situation, see "Handle potential log wrapping" on page 9-15.
Class ID: 30 **Event ID:** 30003	**Class message:** DDR subsystem notification **Specific message:** DDR Log Snooping - Catchup phase completed, userthreads unblocked	3	User transactions are no longer blocked. **Online log:** The specific message also appears in the online log. **ER state:** Active and replicating data. **User action:** None.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 30 **Event ID:** 30004	**Class message:** DDR subsystem failure **Specific message:** WARNING: The replay position was overrun, data may not be replicated.	4	The log replay position was overwritten. **Online log:** The following message appears in the online log: `WARNING: The replay position was overrun, data may not be replicated.` **ER state:** Active and replicating data. Enterprise Replication shuts down if the log read position also gets overwritten. If Enterprise Replication shuts down, event alarm 47 is raised. **User action:** For information about preventing this situation, see "Handle potential log wrapping" on page 9-15.
Class ID: 30 **Event ID:** 30005	**Class message:** DDR Subsystem notification **Specific message:** CDR DDR: Log staging disk space usage reached its allowed configured maximum size *size* (KB). Temporarily disabling log staging.	3	The disk space where logs are stored reached its maximum size. **Online log:** The following message appears in the online log: `CDR DDR: Log staging disk space usage reached its allowed configured maximum size` *`size`* `(KB). Temporarily disabling log staging.` **ER state:** Active and running. Enterprise Replication uses the next configured logical log lag action to protect the replay position. If no other log lag action is configured, the replay position can be overrun. If Enterprise Replication shuts down due to replay position being overrun, restart Enterprise Replication using **cdr cleanstart** command and resynchronize the data. **User action:** Consider increasing the maximum disk space configured for log staging using the CDR_LOG_STAGING_MAXSIZE configuration parameter. The value for the CDR_LOG_STAGING_MAXSIZE configuration parameter can be updated while the server is active using the following command: `onmode -wf CDR_LOG_STAGING_MAXSIZE=`*`size`*

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 30 **Event ID:** 30006	**Class message:** DDR Subsystem notification **Specific message:** CDR: Created staging file *filename* for log unique id *unique_log_id*	3	The log staging file was created. **Online log:** The following message appears in the online log: `CDR: Created staging file filename for log unique id uniqui_log_id` **ER state:** Enterprise Replication is active and staging log files because a log lag state was detected. **User action:** If high-availability secondary servers are configured, consider copying log files to the secondary server. See "Transferring log files to a high-availability cluster secondary server when using ER" on page B-8.
Class ID: 30 **Event ID:** 30007	**Class message:** DDR Subsystem notification **Specific message:** CDR: Completed processing log unique id *unique_log_id*. Deleted log staging file *filename*	2	A log staging file was deleted. **Online log:** The following message appears in the online log: `CDR: Completed processing log unique id unique_log_id. Deleted log staging file filename` **ER state:** Active and replicating data. **User action:** If staged log files are being copied to high-availability secondary servers, consider deleting the log staged log file name specified in the alarm message and the related token log file. See "Transferring log files to a high-availability cluster secondary server when using ER" on page B-8..

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 30 **Event ID:** 30008	**Class message:** DDR Subsystem notification **Specific message:** CDR: Deleted all staging files from log staging directory.	2	The staging files were deleted from the log staging directory. **Online log:** The following message appears in the online log: `CDR: Deleted all staging files from log staging directory.` **ER state:** Active or deleted. Enterprise Replication deletes all files in the log staging directory when they are no longer required. The log files are deleted when any of the following occur: • Enterprise Replication is deleted on the local server. • After the **cdr cleanstart** command is run. • When the value of the LOG_STAGING_DIR configuration parameter is changed (any log files that exist in the previous directory are also deleted). • When Enterprise Replication is defined. **User action:** If staged log files are being manually copied to high-availability secondary server then delete all staged log files on the secondary servers. See "Transferring log files to a high-availability cluster secondary server when using ER" on page B-8.
Class ID: 31 **Event ID:** 31001	**Class message:** ER stable storage pager sbspace is full **Specific message:** CDR Pager: Paging File full: Waiting for additional space in *sbspace_name*	4	This event runs in the foreground. The grouper paging sbspace ran out of space. **ER state:** Active and waiting for the space to be added to the sbspace name specified in alarm-specific message. **User action:** Add a chunk to the specified sbspace. For information about preventing this situation, see "Increasing the Sizes or Numbers of Storage Spaces" on page 9-17.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 31 **Event ID:** 31002	**Class message:** ER stable storage queue sbspace is full **Specific message:** CDR QUEUER: Send Queue space is FULL - waiting for space in *sbspace_name*. CDR QUEUER: Send Queue space is FULL - waiting for space in CDR_QDATA_SBSPACE	4	This event runs in the foreground. The storage space of a queue is full. **Online log:** The specific message also appears in the online log. **ER state:** Active and waiting for space to be added to the sbspace listed. **User action:** Add a chunk to the specified sbspace. If the message specifies CDR_QDATA_SBSPACE, add a chunk to one or more of the sbspaces specified by the CDR_QDATA_SBSPACE configuration parameter. For information about preventing this situation, see "Recovering when Storage Spaces Fill" on page 9-17.
Class ID: 31 **Event ID:** 31003	**Class message:** ER stable storage queue dbspace is full **Specific message:** CDR QUEUER: Send Queue space is FULL - waiting for space in CDR_QHDR_DBSPACE..	4	This event runs in the foreground. The storage space of a queue is full. **Online log:** The specific message also appears in the online log. **ER state:** Active and waiting for space to be added to the dbspace specified by the CDR_DBSPACE configuration parameter. **User action:** Add a chunk to the dbspace specified by the CDR_DBSPACE configuration parameter. For information about preventing this situation, see "Recovering when Storage Spaces Fill" on page 9-17.
Class ID: 32 **Event ID:** 32002	**Class message:** ER: error detected in grouper sub component **Specific message:** CDR Grouper Fanout thread is aborting.	4	The grouper fanout thread is quitting. **ER state:** Enterprise Replication was shut down internally. Event alarm 47 is also raised. **User action:** Restart Enterprise Replication using the **cdr start** command.
Class ID: 32 **Event ID:** 32003	**Class message:** ER: error detected in grouper sub component **Specific message:** CDR Grouper Evaluator thread is aborting.	4	The grouper evaluator thread is quitting. **ER state:** Active and replicating transactions. **User action:** Stop Enterprise Replication with the **cdr stop** command and restart it using the **cdr start** command.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 32 **Event ID:** 32004	**Class message:** ER: error detected in grouper sub component **Specific message:** CDR: Could not copy transaction at log id *log_unique_id* position *log_position*. Skipped.	4	The grouper subcomponent cannot copy the transaction into the send queue. **ER state:** Active and replicating transactions. **User action:** Shut down Enterprise Replication by running the **cdr stop** command, clear the receive queue and restart replication by running the **cdr cleanstart** command, and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 32 **Event ID:** 32005	**Class message:** ER: error detected in grouper sub component **Specific message:** CDR: Paging error detected.	4	The grouper subcomponent detected a paging error. **ER state:** Inactive. **User action:** Restart Enterprise Replication by running the **cdr start** command.
Class ID: 32 **Event ID:** 32006	**Class message:** ER: error detected in grouper sub component **Specific message:** CDR Grouper: Local participant (*participant_name*) stopped for the replicate *replicate_name* (or exclusive replicate set), table (*database*:*owner*.*table*). Data may be out of sync. If replicated column definition was modified then please perform the alter operation at all the replicate participants, remaster the replicate definition then restart the replicate (or exclusive replicate set) definition for the local participant with the data sync option (**-S**).	4	If the grouper subcomponent is not able to convert the replicated row data from the local dictionary format to the master dictionary format, the grouper stops the local participant from the corresponding replicate (or exclusive replicate set) definition and invokes this event alarm.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 32 **Event ID:** 32007	**Class message:** ER: error detected in grouper sub component **Specific message:** CDR *CDR_subcomponent_name*: Could not apply undo properly. SKIPPING TRANSACTION. TX Begin Time: *datetime* TX Restart Log Id: *log_id* TX Restart Log Position: *log_position* TX Commit Time: *datetime* TX End Log Id: *log_id* TX End Log Position: *log_position*	3	The grouper subcomponent did not roll back a transaction to a savepoint. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 33 **Event ID:** 33001	**Class message:** ER: error detected in data sync sub component **Specific message:** Received aborted transaction, no data to spool.	2	Data sync received a transaction that was aborted in the first buffer, so the transaction cannot be spooled to an ATS or RIS file. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 33 **Event ID:** 33002	**Class message:** ER: error detected in data sync sub component **Specific message:** CDR DS *thread_name* thread is aborting.	4	The data sync thread is quitting. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 34 **Event ID:** 34001	**Class message:** ER: error detected in queue management sub component **Specific message:** CDR *CDR_subcomponent_name*: bad replicate ID *replicate_id*	3	This event runs in the foreground. RQM cannot find the replicate in the global catalog for which it has a transaction. **ER state:** Active and replicating transactions. **User action:** Run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 35 **Event ID:** 35001	**Class message:** ER: error detected in global catalog sub component **Specific message:** CDR GC peer request failed: command: *command_string*, error *error_code*, CDR server *CDR_server_ID*	3	Execution of the control command requested by the peer server failed at the local server. **ER state:** Active and replicating transactions. **User action:** Correct the problem identified by the error code. Make sure that the replicate object is the same across all participating servers.
Class ID: 35 **Event ID:** 35002	**Class message:** ER: error detected in global catalog sub component **Specific message:** CDR GC peer processing failed: command: *command_string*, error *error_code*, CDR server *CDR_server_ID*	3	Control command execution at the peer server failed. **ER state:** Active and replicating transactions. **User action:** Correct the problem identified by the error code. Make sure that the replicate object is the same across all participating servers.
Class ID: 35 **Event ID:** 35004	**Class message:** ER: error detected in global catalog sub component **Specific message:** CDR: Could not drop delete table. SQL code *sql_error_code*, ISAM code *isam_error_code*. Table '*database:table*'. Please drop the table manually.	3	The delete table was not dropped while the replicate was being deleted from the local participant. **ER state:** Active and replicating transactions. **User action:** Manually drop the delete table.
Class ID: 36 **Event ID:** 36001	**Class message:** ER: enterprise replication network interface sub component notification **Specific message:** Enterprise Replication: Connection to *servergroupname* closed. Reason: connection request received from an unknown server.	3	Enterprise Replication received a reconnect connection request from an unknown server. **ER state:** Active. **User action:** Check the connection requester server definition in the local server. If the definition is not available on the local server, the remote server definition was probably deleted on the local server by running the **cdr delete server** command, but the **cdr delete server** command was not run on the remote server. In this case, run the **cdr delete server** command on the remote server and, if necessary, redefine the server.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 37 **Event ID:** 37001	**Class message:** ER: error detected while recovering Enterprise Replication **Specific message:** CDR *CDR_subcomponent_name*: bad replicate ID *replicate_id*	3	This event runs in the foreground; Enterprise Replication is blocked until this issue is resolved. **ER state:** Active and replicating transactions. **User action:** If the replicate ID is still valid and exists in **syscdr** catalog tables, run the **cdr check replicateset** command with the **--repair** option to make sure that the data is consistent.
Class ID: 38 **Event ID:** 38001	**Class message:** ER: resource allocation problem detected **Specific message:** CDR *CDR_subcomponent_name* memory allocation failed (*reason*).	2	The specified Enterprise Replication component did not allocate memory. **ER state:** Active. **User action:** Perform these actions: 1. Correct the resource issue. 2. Stop replication by running the **cdr stop** command. 3. Restart replication by running the **cdr start** command. 4. Make sure that the data is consistent by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 39 **Event ID:** 39001	**Class message:** Please notify IBM Informix Technical Support **Specific message:** Log corruption detected or read error occurred while snooping logs.	4	A logical log is corrupted and cannot be processed by the log capture component. Event alarm 47 is also raised in this situation. **Online log:** The following message appears in the online log: `Log corruption detected while snooping logs, logid=log_unique_id logpos=log_position.` **ER state:** Inactive. **User action:** Clear the receive queue and restart replication by running the **cdr cleanstart** command, and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 39 **Event ID:** 39002	**Class message:** Please notify IBM Informix Technical Support **Specific message:** CDR: Unexpected log record type *record_type* for subsystem *subsystem* passed to DDR.	4	A log record of unexpected type was passed to the log capture component. **ER state:** Active and replicating transactions. **User action:** Contact IBM Software Support.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 47 **Event ID:** 47001	**Class message:** CDR is shutting down due to internal error: failure **Specific message:** CDR is shutting down due to internal error: Memory allocation failed	4	Data sync threads encountered a memory allocation error while replaying replicated transactions and replication is stopped. **Online Log:** When the memory allocation error is discovered, the following message appears in the online log: `CDR DS processes is aborting. Signaling CDR system to shutdown as it is low on resources.` When Enterprise Replication is shutting down and the event alarm is being raised, the following message appears in the online log: `CDR is shutting down due to internal error: Memory allocation failed` **ER State:** No replicated transactions are lost while replication is stopped. **User Action:** To resume replication, solve the memory issue and run the **cdr start** command or shut down and restart the database server. If the replay position was overrun while replication was stopped, event alarm 75 is raised.
Class ID: 47 **Event ID:** 47005	**Class message:** CDR is shutting down due to internal error: failure **Specific message:** CDR is shutting down due to an internal error.	4	Enterprise Replication stopped. **ER state:** Inactive. **User action:** Try restarting Enterprise Replication using the **cdr start** command. If replay position overrun is detected and the **cdr start** command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data.
Class ID: 47 **Event ID:** 47006	**Class message:** CDR is shutting down due to internal error: log lag state **Specific message:** CDR DDR: Shutting down ER to avoid a DDRBLOCK situation.	4	Enterprise Replication stopped. **ER state:** Inactive. **User action:** If replay position overrun was detected then restart Enterprise Replication using **cdr cleanstart** command and synchronize the data. If the replay position was not overrun then restart Enterprise Replication using **cdr start** command; there is no need to synchronize the data. If replay position overrun is detected and the **cdr start** command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 48 **Event ID:** 48001	**Class message:** ATS and/or RIS files spooled to disk. **Specific message:** *file name*|*file name*.	3	One or more failed transactions caused the generation of one or more ATS or RIS files. The generated file names are listed in the specific message, separated with a pipe (|) character. **ER State:** Replication is continuing normally. **User Action:** To process the failed transactions, run the **cdr repair** command for each file, or run the **cdr check replicateset** command with the **--repair** option.
Class ID: 49 **Event ID:** 49001	**Class message:** A replication state change event has happened. **Specific message:** Enterprise Replication is started on server *server_name*.	3	This event alarm is disabled by default. The **cdr start** command was run.
Class ID: 50 **Event ID:** 50001	**Class message:** A replication state change event has happened. **Specific message:** Enterprise Replication is stopped on server *server_name*.	3	The **cdr stop** command was run.
Class ID: 51 **Event ID:** 51001	**Class message:** A replication state change event has happened. **Specific message:** Enterprise Replication is suspended on server *server_name*.	3	This event alarm is disabled by default. The **cdr suspend server** command was run.
Class ID: 52 **Event ID:** 52001	**Class message:** A replication state change event has happened. **Specific message:** Enterprise Replication is resumed on server *server_name*.	3	This event alarm is disabled by default. The **cdr resume server** command was run.
Class ID: 53 **Event ID:** 53001	**Class message:** A replication state change event has happened. **Specific message:** Server *server_name* is connected.	3	This event alarm is disabled by default. The **cdr connect server** command was run.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 54 **Event ID:** 54001	**Class message:** A replication state change event has happened. **Specific message:** Server *server_name* is disconnected.	3	This event alarm is disabled by default. The **cdr disconnect server** command was run.
Class ID: 55 **Event ID:** 55001	**Class message:** A replication state change event has happened. **Specific message:** Replication is suspended on replicate *replicate_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr suspend replicate** command was run.
Class ID: 56 **Event ID:** 56001	**Class message:** A replication state change event has happened. **Specific message:** Replication is suspended on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr suspend replicateset** command was run.
Class ID: 57 **Event ID:** 57001	**Class message:** A replication state change event has happened. **Specific message:** Replication is resumed on replicate *replicate_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr resume replicate** command was run.
Class ID: 58 **Event ID:** 58001	**Class message:** A replication state change event has happened. **Specific message:** Replication is resumed on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr resume replicateset** command was run.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 59 **Event ID:** 59001	**Class message:** A replication state change event has happened. **Specific message:** Replication is started on replicate *replicate_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr start replicate** command was run.
Class ID: 60 **Event ID:** 60001	**Class message:** A replication state change event has happened. **Specific message:** Replication is started on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr start replicateset** command was run.
Class ID: 61 **Event ID:** 61001	**Class message:** A replication state change event has happened. **Specific message:** Replication is stopped on replicate *replicate_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr stop replicate** command was run.
Class ID: 62 **Event ID:** 62001	**Class message:** A replication state change event has happened. **Specific message:** Replication is stopped on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr stop replicateset** command was run.
Class ID: 63 **Event ID:** 63001	**Class message:** A replication state change event has happened. **Specific message:** Replication attribute is modified on replicate *replicate_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr modify replicate** command was run.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 64 **Event ID:** 64001	**Class message:** A replication state change event has happened. **Specific message:** Replication attribute is modified on replicate set *replicateset_name* on server *server_name*.	3	This event alarm is disabled by default. The **cdr modify replicateset** command was run.
Class ID: 65 **Event ID:** 65001	**Class message:** A replication state change event has happened. **Specific message:** Change in replicate *replicate_name* on server *server_name*: operation *action*, node[s] *participant_name*.	3	This event alarm is disabled by default. The **cdr change replicate** command was run to add or delete one or more participants.
Class ID: 66 **Event ID:** 66001	**Class message:** A replication state change event has happened. **Specific message:** Change in replicateset *replicateset_name* on server *server_name*: operation *action*, member[s] *replicate_name*.	3	This event alarm is disabled by default. The **cdr change replicateset** command was run to add or delete one or more replicates.
Class ID: 67 **Event ID:** 67001	**Class message:** A replication state change event has happened. **Specific message:** Server *server_name* is deleted.	3	This event alarm is disabled by default. The **cdr delete server** command was run.
Class ID: 68 **Event ID:** 68001	**Class message:** A replication state change event has happened. **Specific message:** Replicate *replicate_name* is deleted on server *server_name*.	3	This event alarm is disabled by default. The **cdr delete replicate** command was run.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 69 **Event ID:** 69001	**Class message:** A replication state change event has happened. **Specific message:** Replicate set *replicateset_name* is deleted on server *server_name*.	3	This event alarm is disabled by default. The **cdr delete replicateset** command was run.
Class ID: 70 **Event ID:** 70001	**Class message:** A replication state change event has happened. **Specific message:** Server *server_name* is modified.	3	This event alarm is disabled by default. The **cdr modify server** command was run.
Class ID: 71 **Event ID:** 71001	**Class message:** ER: Network connection disconnected. **Specific message:** Network connection was dropped from the server *server_name* to the server *server_name*. Connection closed due to an Enterprise Replication administrative activity.	3	The connection was closed as the result of an Enterprise Replication command, such as **cdr stop**, **cdr disconnect server**, or **cdr delete server**. This event alarm appears on the database server on which the command was run and might or might not appear on the peer server. The peer server might receive event alarm 71 with the specific message that the connection closed for an unknown reason because the administrative control message might not reach the peer server before the connection is closed. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name, and that an administrative command was run. **ER State:** How replication is affected and how to reestablish the connection depends on which command closed the connection. • If the **cdr stop** command was run, replicated transactions are no longer being captured from this database server. • If the **cdr disconnect server** command was run, replicated transactions continue to be captured and queued. • If the **cdr delete server** command was run, the database server is no longer a participant in the replication domain and no replicated data is captured on or for this database server. **User Action:** Solve the issue that prompted the running of the administrative command and reestablish the connection between the servers.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 71 **Event ID:** 71002	**Class message:** ER: Network connection disconnected. **Specific message:** Network connection was dropped from the server *server_name* to the server *server_name*. Connection closed due to the idle time-out set for the replication server.	3	An idle timeout occurs when there are no replication messages sent between the replication servers for the number of seconds specified as the idle timeout period. The connection is reestablished automatically when replication messages are ready to be sent. This event alarm appears on both database servers affected by the dropped connection. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name, and the reason of idle timeout. **ER State:** Replication continues normally. **User Action:** None. Replication resumes automatically. You can increase or eliminate the idle timeout period by using the **cdr modify server** command.
Class ID: 71 **Event ID:** 71003	**Class message:** ER: Network connection disconnected. **Specific message:** Network connection was dropped from the server *server_name* to the server *server_name*. Connection unexpectedly closed for an unknown reason.	3	This event alarm occurs when there is a connection problem not related to Enterprise Replication, such as a network outage or one of the database servers shutting down. This event alarm might appear on both database servers affected by the dropped connection. This alarm does not appear on a database server that shut down. This alarm might appear when a peer server closed the connection with an administrative activity, in which case that server receives event alarm 71 with the specific message that an administrative activity closed the connection. **Online Log:** A message appears stating: `CDR connection to server lost`, with the server ID, server name. **ER State:** Replicated transactions continue to be captured and queued, except on database servers that are shut down. Replicated transactions to and from the affected servers are not transmitted. **User Action:** Examine both servers to determine the cause of the dropped connection. • If there was a network problem, solve it and restart any database servers that might be shut down. The Enterprise Replication connection reconnects automatically. • If there was an administrative action, solve the issue that prompted the running of the administrative command and reestablish the connection between the servers.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 73 **Event ID:** 73001	**Class message:** Enterprise replication NIF connection terminated. **Specific message:** Enterprise Replication: Connection to *server_name* closed. Reason: CDR server *server_name* not found.	3	The remote server initiated a connection request that was not completed by the local server. This alarm appears when the remote server has an **sqlhosts** entry for the local server, but the local server does not have a corresponding **sqlhosts** entry for the remote server. This situation can occur when a new server is added to the domain but the local server does not have an entry for that server in its **sqlhosts** file. **Online Log:** The specific message also appears in the online log. **ER State:** The new server cannot participate in replication until the **sqlhosts** entries are correct. Replication between the established replication servers continues normally. **User Action:** To solve this issue, update the **sqlhosts** entry on the local server with the appropriate entry for the remote server. Make sure that all the **sqlhosts** files are consistent on all replication servers in domain.
Class ID: 74 **Event ID:** 74001	**Class message:** Enterprise replication recovery failed **Specific message:** Server name/id mismatch in **sqlhosts** file while recovery, recovered name = *server_name*, id = *ID*, current name = *server_name*, id = *ID*	3	The server information in the **sqlhosts** file was updated after the server was defined for replication. This alarm can appear after the following sequence of events: 1. Replication is stopped on a server because the **cdr stop** command was run or the server was shut down. 2. The **sqlhosts** file was updated. 3. Replication was attempted to be restarted by running the **cdr start** command or by starting the server. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Update the **sqlhosts** file to restore the original server information and then restart replication by running the **cdr start** command or restarting the database server.

Table 9-4. Enterprise Replication Event Alarms (continued)

Class ID and Event ID	Class and Specific Messages	Severity	Explanation
Class ID: 75 **Event ID:** 75001	**Class message:** ER: the logical log replay position is not valid. Restart ER with the **cdr cleanstart** command, and then synchronize the data with the **cdr check --repair** command. **Specific message:** The replay position (logical log ID *log_number* and log position *log_position*) has been overwritten.	4	This event alarm occurs if a database server was shut down or replication was stopped for long enough to fill a logical log. When the database server was restarted or the **cdr start** command was run, replication failed. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Clear the receive queue and restart replication by running the **cdr cleanstart** command and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 75 **Event ID:** 75002	**Class message:** ER: the logical log replay position is not valid. Restart ER with the **cdr cleanstart** command, and then synchronize the data with the **cdr check --repair** command. **Specific message:** The replay position (logical log ID *log_number* and log position *log_position*) is later than the current position.	4	This event alarm can occur after a point in time restore was performed on the database server. The point in time restore applied log records beyond the current replay position. **Online Log:** The specific message also appears in the online log. **ER State:** Replication is stopped on this server. **User Action:** Clear the receive queue and restart replication by running the **cdr cleanstart** command and then synchronize the data by running the **cdr check replicateset** command with the **--repair** option.
Class ID: 76 **Event ID:** 76001	**Class message:** A replication state change event has happened. **Specific message:** Server *server_name* is disabled.	3	This event alarm is disabled by default. The **cdr disable server** command was run.
Class ID: 77 **Event ID:** 77001	**Class message:** A replication state change event has happened. **Specific message:** Server *server_name* is enabled.	3	This event alarm is disabled by default. The **cdr enable server** command or the **cdr check replicateset** command with the **--enable** option was run.

Related concepts:

"Monitor Enterprise Replication" on page 9-1

Related reference:

"cdr delete replicate" on page A-79
"cdr delete replicateset" on page A-80
"cdr delete server" on page A-81
"cdr change replicate" on page A-32
"cdr change replicateset" on page A-35
"cdr connect server" on page A-57
"cdr start" on page A-129
"cdr stop" on page A-150
"cdr suspend server" on page A-158
"cdr resume server" on page A-128
"cdr delete template" on page A-84
"cdr suspend replicate" on page A-155
"cdr suspend replicateset" on page A-157
"cdr resume replicate" on page A-126
"cdr resume replicateset" on page A-127
"cdr start replicate" on page A-131
"cdr start replicateset" on page A-134
"cdr stop replicate" on page A-152
"cdr stop replicateset" on page A-154
"cdr modify replicate" on page A-108
"cdr modify replicateset" on page A-111
"cdr modify server" on page A-112

Enabling or Disabling Enterprise Replication Event Alarms

You can control which Enterprise Replication event alarms can be raised.

By default, Enterprise Replication event alarms are enabled, except most of the state change event alarms that are raised by specific **cdr** commands. You might want to enable state change event alarms to track which **cdr** commands are being run, but if you are setting up your replication system and running many **cdr** commands, the resulting large number of event alarms might affect system performance.

To change which Enterprise Replication event alarms are enabled, reset the values of the CDR_ENV configuration parameter for the CDR_ALARMS environment variable:

1. Add a new line or update the existing line for CDR_ENV CDR_ALARMS in the `onconfig` file. List all the Enterprise Replication event alarms that you want to be enabled.
2. Restart the database server.

Example

The following example shows the CDR_ENV value in the `onconfig` file for the CDR_ALARMS environment variable with event alarm 49 enabled in addition to the default event alarms:

```
CDR_ENV CDR_ALARMS=30-39,47-50,71,73-75
```

Related reference:

"CDR_ALARMS Environment Variable" on page B-15
"CDR_ENV Configuration Parameter" on page B-3

Part 3. Appendixes

Appendix A. The cdr Command-Line Utility Reference

The **cdr** command-line utility (CLU) lets you configure and control Enterprise Replication from the command line on your UNIX or Windows operating system.

Related tasks:

"Repairing Failed Transactions with ATS and RIS Files" on page 8-21

Interpreting the cdr Command-Line Utility Syntax

This topic defines the terminology and conventions used in the descriptions of the **cdr** command-line utility (CLU).

Each **cdr** 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 to execute any of the CLU commands except the **cdr list** commands, unless otherwise noted.

You can run **cdr** commands from within SQL statements by using the SQL administration API. Most **cdr** commands that perform actions are supported by the SQL administration API; **cdr** commands that display information are not supported. For more information, see the *IBM Informix Administrator's Reference*.

Related concepts:

"Enterprise Replication Server Administrator" on page 2-1

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
```

Command abbreviations are not allowed when you run **cdr** commands within SQL statements using the SQL administration API. For more information, see the *IBM Informix Administrator's Reference*.

Option Abbreviations

Each option for a **cdr** 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.

On UNIX, a long form example might look like:

```
cdr define server --connect=ohio --idle=500 \
   --ats=/cdr/ats --initial utah
```

On WINDOWS, the same long form example would 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
```

WINDOWS:

```
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 usually the first letter of the long form. The short form never requires an equal sign. However, sometimes the short form is capitalized and sometimes it is not. To find the correct syntax for the short form, check the table that accompanies each command variation.

Tip: 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-63, 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 **cdr** commands in any order. Some of the syntax diagrams 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, if you duplicate an option you will receive 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: For ease of maintenance, 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.

On UNIX, the command line might look like:

```
cdr define server --connect=katmandu --idle=500 \
   --ats=/cdrfiles/ats

cdr def ser -c katmandu -i 500 -A /cdrfiles/ats
```

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

Identifier names used in **cdr** commands follow the guidelines of SQL syntax.

Identifiers are the names of objects, such as database servers, databases, columns, replicates, replicate sets, and so on, that Informix and Enterprise Replication use.

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 IBM Informix, 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 *IBM Informix Guide to SQL: Syntax*.

The length of a path and file name, such as the names of 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

Most **cdr** commands allow a connect option to specify the database server to connect to for performing the command.

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.

Connect Option:

Element	Purpose	Restrictions	Syntax
server	Name of the database server to connect to	The name must be the name of a database server or server connection. The `sqlhosts` entry for the server must not include the security option **s=6** unless an encrypted password file is present. .	"Long Identifiers" on page A-3
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. The `sqlhosts` entry for the server name must not include the security option **s=6** unless an encrypted password file is present. .	"Long Identifiers" on page A-3

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.

If your replication domain contains database servers that are running different server versions, **cdr** commands must connect to the server running the latest version of IBM Informix. If you are connected to a database server running an older version of IBM Informix, you cannot run a **cdr** command on a database server running a later version of IBM Informix.

If the database server uses trusted connections between replication servers by including the **s=6** option in the `sqlhosts` entries, you configure a regular connection to an alias of the server for the **cdr** utility to use. In a trusted connection environment, the **cdr** utility can only connect to the local replication server.

Related reference:

"Global Catalog" on page 2-3

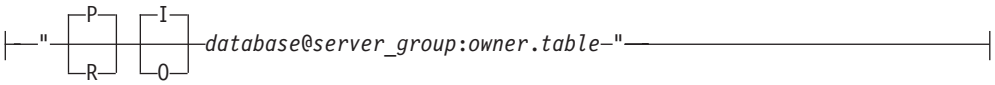
The onpassword utility (Administrator's Reference)

Participant and participant modifier

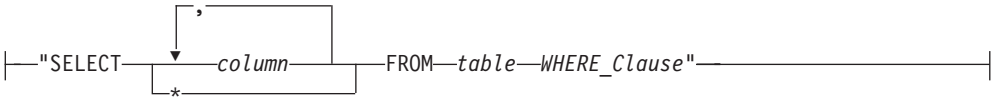
A participant defines the data (database, table, and columns) to be replicated on a specific database server. You can choose whether to allow the participant to both send and receive replicated data, or to just receive replicated data. You can choose to check for table owner permissions when applying operations. By default, permissions are not checked. The participant modifier is a restricted SELECT statement that specifies the rows and columns that will be replicated.

Syntax

Participant:



Participant Modifier:



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-3
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-3
owner	User ID of the owner of the table to be replicated.		"Long Identifiers" on page A-3
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 and must be used only once in the same replicate.	"Long Identifiers" on page A-3
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-3
WHERE_Clause	Clause that specifies a subset of table rows to be replicated.		WHERE clause syntax

The following table describes the participant options.

Option	Meaning
I	Default. Disables the table-owner option (**O**).

Option	Meaning
O	Enables permission checks for table owner specified in the participant 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**. On 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. On Windows, if a trigger requires any system-level commands, the system-level commands are executed as a less privileged user because you cannot impersonate another user without having the password, whether or not the participant includes the **O** option.
P	For primary-target replicates, specifies that the participant is a primary participant, which both sends and receives replicated data. Do not use for an update-anywhere replicate. Enterprise Replication defines all the participant as primary in an update-anywhere replication system.
R	For primary-target replicates, specifies that the participant is a receive-only target participant, which only receives data from primary participants.

Usage

Each participant in a replicate must specify a different database server. The participant definition includes the following information:

- Database in which the table resides
- Table name
- Table owner
- Participant type
- Participant modifier with a SELECT statement

You must include the server group, database, table owner, and table name when defining a participant, and enclose the entire participant definition in quotation marks.

Restriction: 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**
- **cdr define template**

The participant modifier must be enclosed in quotation marks.

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.

Only replicate between like data types. For example, do not replicate between the following:

- CHAR(40) to CHAR(20)
- INT to FLOAT

You can replicate between the following types with caution:

- SERIAL and INT
- BYTE and TEXT
- BLOB and CLOB

Note: The ERKEY shadow columns are not included in the participant definition if you use SELECT * in your participant modifier. To include the ERKEY shadow columns in the participant definition, use the **--erkey** option with the **cdr define replicate**, **cdr change replicate**, or **cdr remaster** commands.

Examples

Example 1: Defining update-anywhere participants

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"
```

Example 2: Defining a primary server

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 sent to the secondary servers.

Example 3: Defining a server that only receives data

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

```
"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.

The central server, **london**, is a standard replication server without restrictions on sending or receiving data.

Related concepts:

"Primary-Target Replication System" on page 3-1

"Considerations for Replicating Opaque Data Types" on page 2-17

"Participant definitions" on page 6-7

Related reference:

"cdr define replicate" on page A-61

"cdr modify replicate" on page A-108

"cdr change replicate" on page A-32

"cdr define template" on page A-75

"cdr swap shadow" on page A-159

Return Codes for the cdr Utility

If a **cdr** command encounters an error, the database server returns an error message and a return code value.

The message briefly describes the error. For information about interpreting the return code, use the **cdr finderr** command.

The following table lists the return codes.

Table A-1. Return codes for the cdr utility

Return code	Error text	Explanation
0	Command successful.	
1	A connection does not yet exist for the given server.	A replication server involved in the command is not connected to the server that is running the command. This error code can be returned when a **cdr sync** or **cdr check** task cannot switch connections between task participants. **User action:** Establish connections between all necessary replication servers and rerun the command.
3	Table column undefined.	A column name listed in the SELECT statement of the replicate participant definition is not found in the table dictionary. This error code can be returned if a shadow column name is included in the SELECT statement of the replicate definition. **User action:** Correct the SELECT statement of the participant definition.
4	Incompatible server version.	A **cdr** command originating on a database server running an older version of Informix attempted to run on a database server running a later version of Informix. **User action:** Run the command from the replication server running the most recent version of Informix.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
5	Unable to connect to server specified.	A replication server involved in the command is not available for one of the following reasons: • The server disconnected from the domain. • Replication is no longer active on the server. • The server is offline. • The **--connect** option was not used and the **INFORMIXDIR** environment variable for the current server is not set. This error code can be returned if one of the **cdr sync** or **cdr check** task participants cannot be accessed or if a task participant became inactive or went offline while a sync or check task is in progress. This error code can be returned if the user running the **cdr define replicate** or **cdr change replicate** command does not have Connect privilege on the database specified for the replicated table. **User action:** Check the status of all participating servers and rerun the command when all servers are active.
6	Database does not exist.	The database name specified for the replicate in the command does not exist. **User action:** Verify the spelling of the database names and that they exist on each participant and rerun the command. This error code can be returned if the **cdr view** command is run and the **sysadmin** database does not exist.
7	Database not logged.	The database specified for the replicate in the command is a non-logging database. Replicated databases must be logged. **User action:** Change the database logging mode to buffered logging and rerun the command.
8	Invalid or mismatched frequency attributes.	The value for the **--at** or **--every** option is not within the range of valid values or is formatted incorrectly. **User action:** Rerun the command with valid and correctly formatted frequency values.
9	A connection already exists for the given server.	This error code can be returned if the **cdr connect server** command is run for a server that already has an active connection.
10	Invalid replicate set state change.	The replicate set specified in the command is not in the appropriate state for the command. This error code is returned in the following situations: • The **cdr stop replicateset** command is run but all replicates in the replicate set are not active. • The **cdr start replicateset** command is run but all replicates in the replicate set are already active. • The **cdr suspend replicateset** command is run but all replicates in the replicate set are not active. • The **cdr resume replicateset** command is run but all replicates in the replicate set are already active. **User action:** Run the **cdr list replicateset** and **cdr list replicate** commands to see the status of each replicate.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
11	Undefined replicate set.	The specified replicate set does not exist or the replicate set is empty. The replicate set name might be incorrectly specified in the command. **User action:** Rerun the command with the correct replicate set name, or add replicates to the replicate set and then rerun the command.
12	Replicate set name already in use.	The replicate set name specified in the command is already being used. Replicate set names must be unique in the domain. **User action:** Run the **cdr list replicateset** command to view a list of replicate set names and then rerun the original command with a different replicate set name.
13	Invalid idle time specification.	The value for the **--idle** option is not within the range of valid values or is formatted incorrectly. **User action:** Rerun the command with a valid and correctly formatted value.
14	Invalid operator or specifier.	Both the **--ignoredel y** and the **deletewins** options were used in the same command. These options cannot be used together. **User action:** Rerun the command with only one of these options.
15	Invalid length.	The ATS or RIS directory path specified in the command exceeds 256 characters. This error can be returned if the server group name exceeds 127 characters. **User action:** Rerun the command with a shorter directory path or server group name.
16	Replicate is not a member of the replicate set.	The replicate specified to be deleted from the replicate set is not a member of the replicate set. **User action:** Run the **cdr list replicateset** command for the replicate set to view a list of replicates in the replicate set and then rerun the original command with the correct replicate name.
17	Participants required for operation specified.	One or more of the participants necessary for this command were not specified. This error code is returned if the sync source server or the target server is not defined as a participant for the **cdr sync** or **cdr check** task. This error code is also returned if the target participant list is empty. **User action:** Rerun the command with the required participants.
18	Table does not contain primary key.	The table specified in the command does not have a primary key. This error is returned if the **cdr sync** or **cdr check** task cannot find the primary key for the table being repaired. **User action:** Add a primary key constraint to the table and rerun the command.
19	Table does not exist.	The table owner name specified in the command is not correct. This error is also returned if the table owner name is not specified for a table in an ANSI database. **User action:** Rerun the command with the correct table owner name.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
20	Server already participating in replicate.	The participant specified in the command is already a participant in the replicate.
22	Primary key not contained in select clause.	The replicate participant SELECT statement did not include the primary key column. **User action:** Rerun the command including the primary key column in the participant SELECT statement.
25	Replicate already participating in a replicate set.	The replicate specified to be added to the replicate set is already a member of the replicate set. **User action:** Run the **cdr list replicateset** command to view a list of replicates in the replicate set.
26	Replicate set operation not permitted on replicate.	The replicate specified to be deleted from the replicate set does not have a valid name. **User action:** Run the **cdr list replicateset** command to view a list of replicates in the replicate set and then rerun the original command with the correct replicate name.
28	Replicate name already in use.	The replicate name specified in the command is already being used. Replicate names must be unique in the domain. **User action:** Run the **cdr list replicate** command to view a list of replicate names and then rerun the original command with a different replicate name.
29	Table does not exist .	The table name specified in the command does not exist. **User action:** Rerun the command with an existing table name.
30	Invalid replicate state change.	The replicate specified in the command is not in the appropriate state for the command. This error code is returned in the following situations: • The **cdr stop replicate** command is run but the replicate is not active. • The **cdr start replicate** command is run but the replicate is already active. • The **cdr suspend replicate** command is run but the replicate is not active. • The **cdr resume replicate** command is run but the replicate is already active. **User action:** Run the **cdr list replicate** command to see the status of the replicate.
31	Undefined replicate.	The replicate name cannot be found in Enterprise Replication catalog tables. The name of the replicate might be incorrectly specified in the command. **User action:** Rerun the command with the correct replicate name.
32	sbspace specified for the send/receive queue does not exist	The CDR_QDATA_SBSPACE configuration parameter is not set to a valid sbspace name. **User action:** Set the CDR_QDATA_SBSPACE configuration parameter to a valid sbspace name in the `onconfig` file.
33	Server not participant in replicate/replicate set.	The server name specified in the command is not a participant in the replicate or replicate set. This error is returned if a server name is not valid. **User action:** To see a list of all participants for each replicate, query the **syscdrpart** view in the **sysmaster** database.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
35	Server not defined in sqlhosts.	The server group name specified in the command is not defined in the `sqlhosts` file specified by the **INFORMIXSQLHOSTS** environment variable. **User action:** Check the `sqlhosts` file for the correct spelling of the server group name, or, if necessary, update the `sqlhosts` file to add the server group, and then rerun the original command.
37	Undefined server.	The target participant cannot be found in the Enterprise Replication catalog tables. The name of the server might be incorrectly specified in the command. **User action:** Rerun the command with the correct server name.
38	SPL routine does not exist.	The SPL routine specified with the **--conflict** option does not exist on one or more participants. **User action:** Make sure that the SPL routine exists on all participants and rerun the command.
39	Invalid select syntax.	The SELECT statement included in the command is not valid or is missing from the command. **User action:** Rerun the command with the correct SELECT statement.
40	Unsupported SQL syntax (join, etc.).	The SELECT statement contains syntax that is not supported for replicate participants. Syntax such as subqueries in the WHERE clause or selecting from multiple tables with a JOIN clause is not supported. **User action:** Rerun the command with the correct SELECT statement.
41	GLS files required for data conversion are not installed.	The GLS files required for data conversion to or from UTF-8 are not installed. Code set conversion files are installed with the Client SDK and are in the **$INFORMIXDIR/gls/cv9** directory.
42	Invalid time range.	The time range does not have valid values or is formatted incorrectly. **User action:** Rerun the command with a valid and correctly formatted time range.
43	Participants required for specified operation.	The command did not include the required participant information. **User action:** Rerun the command with participant information.
44	Invalid name syntax.	The name of a replicate or server in the command is not valid, for example, the name might be too long. **User action:** Rerun the command with a valid name.
45	Invalid participant.	The participant syntax is not valid. **User action:** Rerun the command with a valid participant syntax.
47	Invalid server.	A connection between the current server and the specified server is not allowed. This error code is returned if a server attempts to connect to a leaf server that has a different parent server. This error code is also returned if the server specified in the **cdr repair** command does not exist in the Enterprise Replication catalog.
48	Out of memory.	Enterprise Replication cannot allocate enough memory for this command.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
49	Maximum number of replicates exceeded.	The maximum number of replicates that can be defined from a particular server is exceeded. **User action:** Rerun the command while connected to a peer replication server.
52	Server name already in use.	A replication server with this group ID exists. **User action:** Run the **cdr list server** command to see a list of all replication server names and group IDs
53	Duplicate server or replicate.	A replication server or replicate name is listed more than once in the command. This error code is returned if the sync source server is also specified as a sync target server or if the same server is listed multiple times as a sync target participant. This error code is returned if the same group name is defined more than once in the `sqlhosts` file. **User action:** Rerun the command specifying each server and replicate one time.
54	Invalid conflict rule specified.	The conflict resolution rule is not correctly specified. This error code is returned for the **cdr define replicate** or **cdr modify replicate** command under the following circumstances: • A stored procedure is specified as the conflict resolution rule but the table has user-defined data types or collection data types. • A secondary conflict resolution rule is specified that is not a stored procedure conflict resolution rule. • A secondary conflict resolution rule is specified but the primary conflict resolution rule is not time stamp or delete wins. This error code is returned if the **--timestamp** option is used in a **cdr check** command and the replicate specified in the command does not use the time stamp or the delete wins conflict resolution rule. This error code is also returned when the **cdr check** command includes the **--deletewins** option but the specified replicate does not use the delete wins conflict resolution rule. **User action:** Correct the conflict resolution rule issue and rerun the command.
55	Resolution scope not specified.	The conflict resolution scope (row or transaction) is required for ER to resolve conflicts between replicated transactions. Scope is not required if the conflict resolution rule is **ignore**, in which case ER does not attempt to resolve conflicts. **User action:** Rerun the command with a conflict resolution scope.
56	Shadow columns do not exist for table.	A conflict resolution rule requires the **cdrtime** and **cdrserver** shadow columns but those columns do not exist in the replicated table. **User action:** Alter the replicated table to add the shadow columns by using the ADD CRCOLS clause and rerun the original command.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
57	Error creating delete table.	The delete table corresponding to the replicated table was not created. **User action:** Check the server message log file for additional error messages.
58	No conflict resolution rule specified.	A conflict resolution rule was not specified in the command. **User action:** Rerun the command with the **--conflict** option to specify a conflict resolution rule.
61	User does not have permission to issue command.	The user running this command does not have the DBSA privilege at one of the participants in the command. **User action:** Acquire the DBSA privilege on all participants and rerun the command, or rerun the command as a user that has the DBSA privilege at all participants.
62	Enterprise Replication not active.	The command cannot run because Enterprise Replication is not active on the server. **User action:** Run the **cdr list server** command to see the status of the server.
63	Enterprise Replication already active.	The command cannot make Enterprise Replication active because ER is already active on the server. **User action:** Run the **cdr list server** command to see the status of the server.
64	Remote/cyclic synchronization not allowed.	The command to define a replication server was attempted on a remote server. **User action:** Rerun the command on the server that is being defined.
65	Server identifier already in use.	The server group ID is not unique. **User action:** Rerun the command with a unique server group ID.
66	No upper time for prune error.	The ending date value for the error pruning range was not specified. **User action:** Rerun the command with a valid ending date.
67	Error not found for delete or update.	The error sequence number does not exist in the errors table. **User action:** Run the **cdr error** command to see a list of error sequence numbers and then rerun the command with an existing number.
68	Invalid participant mode.	The participant type value is not valid. **User action:** Rerun the command with a valid participant type.
69	Conflict mode for replicate not ignore or always apply.	One or more replicate participants specified in the command is defined as receive-only and must use a conflict resolution rule of **ignore** or **always**. **User action:** Rerun the command with the **--conflict** option set to **ignore** or **always**.
70	Connect/disconnect to/from same server.	The command attempted to connect the local server to itself. **User action:** Rerun the command with a different server name.
72	Cannot delete server with children.	The command did not delete the hub server because the hub server still has child servers. **User action:** Delete the child servers and then delete the hub server.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
75	Request denied on limited server.	The command is not allowed on leaf servers. It is also not allowed on replication servers that are disabled. **User action:** If the server is in disabled mode, wait until the server is active and rerun the command.
77	Could not drop the Enterprise Replication database.	The **syscdr** database was not deleted because a client is accessing it. **User action:** Wait for the client to unlock the **syscdr** database and then rerun the command. If necessary, use the **--force** option to drop the **syscdr** database if Enterprise Replication was partially deleted.
78	Invalid ATS directory.	The ATS directory path specified in the command was not valid for one of the following reasons: • The path does not exist. • The path is not a directory. • The path is /dev/null (UNIX) or NUL (Windows). **User action:** Rerun the command with a valid ATS directory path.
79	Invalid RIS directory.	The RIS directory path specified in the command was not valid for one of the following reasons: • The path does not exist. • The path is not a directory. • The path is /dev/null (UNIX) or NUL (Windows). **User action:** Rerun the command with a valid RIS directory path.
80	Invalid conflict resolution change.	The conflict resolution rule of a replicate cannot be changed to **ignore** or from **ignore**.
84	No sync server.	A synchronization server must be specified if the replication server being defined is a non-root or leaf server. The first server in a replication domain must be a root server. **User action:** Rerun the command with the **--sync** option.
85	Incorrect participant flags.	The participant type or owner option included in the command was not valid. **User action:** Rerun the command with valid participant options.
86	Conflicting leaf server flags.	The **--nonroot** and **--leaf** options cannot be used together. **User action:** Rerun the command with only one of the options.
90	CDR connection to server lost, id *group_id*, name *groupname*. Reason: System clocks off by %d seconds.	The system clock times on the servers differ by more than 900 seconds.
91	Invalid server state change.	The server is already in the state indicated by the command. This error code can be returned if the **cdr suspend server** command is run on a server that is suspended or if the **cdr resume server** command is run on a server that is active. **User action:** Run the **cdr list server** command to see the status of the server.
92	CDR is already defined.	Enterprise Replication is already defined on this server.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
93	Enterprise Replication is currently initializing.	Enterprise Replication cannot be stopped on the server because replication is in the process of being initialized. **User action:** Run the **cdr list server** command to see the status of the server. Rerun the command when replication is active.
94	Enterprise Replication is currently shutting down.	Enterprise Replication cannot be stopped on the server because replication is already in the process of being stopped. **User action:** Run the **cdr list server** command to see the status of the server. If necessary, rerun the command.
99	Invalid options or arguments passed to command.	One or more of the options included with this command are not valid options. **User action:** Rerun the command with valid options.
100	Fatal server error.	The command was not completed because of an unrecoverable error condition.
101	This feature of Enterprise Replication is not yet supported.	One of the participants included with this command is running a version of Informix that does not support this command. **User action:** Rerun the command with valid options.
102	Root server cannot sync with non root or leaf servers.	The synchronization server must be a root server. The **--sync** option cannot specify a non-root or leaf server. **User action:** Rerun the command specifying a root server with the **--sync** option.
103	Invalid server to connect.	A non-root server can connect only to its parent or children servers. **User action:** Rerun the command specifying to connect to the parent or a child server.
105	UDR needed for replication was not found.	A user-defined type listed in the SELECT statement of the participant definition does not have one or more of the **streamread()**, **streamwrite()**, or **compare()** support routines. **User action:** Create the required routines for the user-defined type and rerun the command.
106	Setup necessary for UDR invocation could not be completed.	The set-up process necessary to run the **streamread()**, **streamwrite()**, or **compare()** routine for a user-defined type included in the participant definition failed. **User action:** Check to be sure that the required routines for the user-defined type exist. Create them if necessary and rerun the command.
107	Sbspace specified for the send/receive queue does not exist.	The sbspace specified for the CDR_QDATA_SBSPACE configuration parameter is not a valid name or does not exist. **User action:** Correct the value of the CDR_QDATA_SBSPACE configuration parameter or create the sbspace and rerun the command.
110	Data types with out of row or multi-representational data are not allowed in a replicate where clause.	A replicate participant WHERE clause cannot include a data type that has out-of-row data, such as, a collection data type, a user-defined type, or a large object type. **User action:** Remove the column with out-of-row data from the participant WHERE clause and rerun the command.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
111	Cannot have Full Rows off and use stored procedure conflict resolution.	The stored procedure conflict resolution rule requires full row replication. **User action:** Rerun the command without the **--fullrow=n** option.
112	The replicate set command could only be partially executed. Please run **cdr list replset ReplSetName** to check results.	The command was successful on some, but not all, of the replicates in the replicate set. **User action:** Run the **cdr list replicate** command to see the status of each replicate and run the appropriate command on each of the remaining replicates.
113	Exclusive Replset violation.	The specified replicate is a member of an exclusive replicate set, which requires this operation to be performed for the replicate set instead of for individual replicates. **User action:** Run the equivalent command for the replicate set.
115	The syscdr database is missing.	The **syscdr** database cannot be opened. **User action:** Check server message log file for any additional error messages. If you received this error code after running the **cdr delete server --force** command, no action is required on the server being deleted. Run the **cdr delete server** command to delete that server on all peer replication servers in the domain. If you receive this error after running the **cdr start** command, make sure that Enterprise Replication is defined on the local server, and if necessary, define it by running the **cdr define server** command.
116	Dbspace indicated by CDR_DBSPACE is invalid.	The dbspace specified as the value of the CDR_DBSPACE configuration parameter does not exist. **User action:** Correct the value of the CDR_DBSPACE configuration parameter or create the dbspace and rerun the command.
117	Enterprise Replication operation attempted on HDR secondary server.	Enterprise Replication commands are not valid on high-availability secondary servers. **User action:** Rerun the command on a high-availability primary server.
118	SQLHOSTS file has multiple entries either at group ID or group name.	There are multiple group definitions for the same group name in the `sqlhosts` file. **User action:** Update the `sqlhosts` file to make all group entries unique.
119	SQLHOSTS file has a problem with (g=) or (i=) option.	The group name specified in the command is not found in the `sqlhosts` file. **User action:** Rerun the command with a valid group name or update the `sqlhosts` file and then rerun the command.
120	Cannot execute this command while ER is active.	Enterprise Replication cannot be deleted on this server because replication is still active. **User action:** Run the **cdr stop** command and then rerun the **cdr delete server --force** command.
121	Master participant not found.	The replication server that is specified as the master server in the command does not exist or is not a participant in the specified replicate. **User action:** Rerun the command with the correct master server name.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
122	Attempt to perform invalid operation including shadow replicates.	The replicate specified in the command has shadow replicates, which prevent the command from completing. **User action:** Run the **cdr list replicate** command to see shadow replicate information. Wait for the shadow replicate to be deleted and then rerun the original command. If you are deleting the replicate, delete the shadow replicate and then rerun the original command.
123	Attempt to include an invalid participant in a shadow replicate.	The command attempted to add a participant to a shadow replicate that does not exist in the primary replicate. **User action:** Rerun the command with a valid participant.
124	Invalid command passed to cdrcmd function.	An argument that is not valid was passed to an internal routine. **User action:** Contact IBM Software Support.
125	An error occurred concerning a mastered replicate.	The server specified as the master server in the command is not included as a participant in the replicate. This error code is returned if the mastered dictionary verification fails when adding a participant to a mastered replicate. This error code is returned if Enterprise Replication encounters an internal error during master replicate definition. **User action:** Rerun the command with the master server included in the participant list or check the table dictionary.
126	Invalid template participant.	The same table name was specified more than once in the **cdr define template** command, or a participant name in the **cdr realize template** command is not valid. **User action:** Rerun the command with unique table names or with valid participant names.
127	Template name already in use.	A replication template with this name exists. **User action:** Rerun the command with a unique template name.
128	Undefined template.	The template name specified in the command does not exist. **User action:** Rerun the command with an existing template name.
129	Cannot delete specified replset as it is a template.	The replicate set specified in the command is a part of a template and cannot be deleted with this command. **User action:** Run the **cdr delete template** command to delete a template.
131	Sync server not specified.	The synchronization server specified in the command must be the same server that was specified in the **cdr define repair** command. **User action:** Rerun the command with the correct synchronization server.
132	Invalid sync server specified. Server is not yet defined in ER topology.	The synchronization server specified in the command is not a replication server. **User action:** Rerun the command with an existing replication server as the synchronization server.
134	Cannot lock the replicated table in exclusive mode. For more information see message log file.	The command cannot obtain an exclusive lock on the table to set alter mode. **User action:** See the online message log file for other errors.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
135	Replicate/table is not in alter mode.	The table specified in the command is not in alter mode and therefore alter mode cannot be turned off.
136	Snoopy sub-component is down.	Alter mode cannot be set because the log capture thread was not active.
137	Mismatch between local table dictionary and master dictionary.	The master dictionary does not match the local participant dictionary. **User action:** Check the replicated table definitions on all participants.
138	Replicates not found for table. For more information see message log file.	Alter mode was not turned off because the replicate definitions for the specified table cannot be found. **User action:** Check the spelling of the table name and rerun the command.
139	Mismatch in replicate names or states. Primary and shadow replicate states must match. See the message log file for more information.	The primary and shadow replicates are not in the same state. **User action:** Run the **cdr list replicate** command to see the replicate state. When the primary and shadow replicates have the same state, rerun the original command.
140	Primary and shadow replicate participant verification failure.	The primary and shadow replicate information does not match.
141	Table is already in alter mode. For more information see message log file.	Alter mode cannot be turned on because the table is already in alter mode.
142	Classic replicates (no mastered dictionary) defined on the table. See message log file for more information.	One or more non-mastered replicates are defined on the specified table. Alter mode requires mastered replicates.
146	Resynchronize error, job name is already in use.	The job name must be unique. **User action:** Rerun the command with a unique job name.
147	Resynchronize error, specified replicate is a shadow repl.	The replicate specified in the command is a shadow replicate. The operation cannot be performed on a shadow replicate. **User action:** Run the **cdr list replicate** command to see a list of replicate names and rerun the original command with a primary replicate.
148	Only either participant list or target server can be given for a define repair command.	Both the target server name and a participant list were included in the command. **User action:** Rerun the command with either a target server name or a participant list.
151	Resynch job can be started or stopped only at the source server.	This command must be run from the server specified as the synchronization data source. **User action:** Rerun the command while connected to the synchronization data source server.
154	The replicate being repaired must be in active state.	The replicate specified in the command cannot be repaired because it is not active. **User action:** Run the **cdr list replicate** command to see the states of replicates.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
156	Cannot perform auto remastering process. Replicate is not defined with column name verification option (–name=y). Perform manual remastering process.	Automatic remastering is not possible for the specified replicate. **User action:** Manually remaster the replicate. For instructions, see "Manual Remastering" on page 8-27.
157	CDR: Cannot verify/block grouper evaluation blocking condition.	The specified table cannot be set in alter mode because the grouper component is not active. **User action:** Run the **onstat -g grp** and **onstat -g ddr** commands to check the status of the grouper and log capture.
158	CDR: Cannot unblock grouper evaluation.	Alter mode cannot be turned off for the table because the grouper component is not active. **User action:** Run the **onstat -g grp** and **onstat -g ddr** commands to check the status of the grouper and log capture.
159	CDR: Grouper evaluation was already blocked in the same transaction. Commit the previous alter statement then re-execute the current alter statement.	More than one alter statement for replicated tables was included in a single transaction. **User action:** Rerun each alter statement in its own transaction.
160	The specified table was not found in the database. The table specified is either a view or an internally created cdr system table and replicate cannot be defined on views and internally created cdr system tables.	A table name specified in the command was not found or is not a type of table that can be replicated. **User action:** Rerun the command with valid table names.
161	Specified file to read table participants *filename* could not be opened. Please check. Template could not be defined.	The file name specified in the command does not exist. **User action:** If necessary, create the file. Rerun the command with the correct file path and name for the table list.
162	CDR: Local group name not defined in ATS/RIS file.	The ATS or RIS file content is not in the correct format. The file might be corrupted.
163	Error detected while checking replicate attributes on the given table.	The specified table cannot be set to alter mode because the table does not have any master replicates defined. Alter mode requires master replicates. **User action:** Run the **cdr list replicate** command to see the replicates that are defined for the table.
164	Cannot repair - ATS/RIS repair failed.	The ATS or RIS file content is not in the correct format. The file might be corrupted.
165	Cannot suspend replicate/replset because of dependent repair jobs.	The replicate or replicate set cannot be suspended until the active repair jobs are complete. **User action:** Wait for the repair jobs to complete. Run the **cdr list replicate** command to see if the shadow replicates associated with the repair jobs still exist. After the shadow replicates are automatically deleted, rerun the original command.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
166	Replicate set does not have any replicates.	The replicate set specified in the commands does not contain replicates. This error code is also returned when no replicates are found for a **cdr check** repair task when the **--allrepl** option is used. **User action:** Run the **cdr list replicateset** command for the replicate set to see its replicates.
167	Enterprise Replication is not supported in Informix Express Edition server.	Enterprise Replication commands cannot be run on servers running Informix Express Edition.
168	The specified table is actually a view, replicate definition on view is not supported.	Replicates cannot be defined on views. **User action:** Rerun the command specifying standard table names.
169	Cannot realize an empty template/	The template cannot be instantiated because it does not contain any replicates. **User action:** Run the **cdr list template** command to see if the template contains replicates.
170	Template is not yet defined that does not have any replicates.	The template cannot be instantiated because it is not defined. **User action:** Run the **cdr list template** command to see the names of defined templates.
171	Classic replicates do not support --verify (-v) and/or --autocreate (-u) options.	The **--verify** and **--autocreate** options are valid only for master replicates. **User action:** Verify the replicate definition by running the **cdr list replicate** command.
172	Checksum libraries not installed.	Enterprise Replication cannot find checksum UDR routines for the **cdr check** command. This error occurs if the replication server is running a version of Informix that does not support the **cdr check replicate** or **cdr check replicateset** command. **User action:** If the replication server is running Informix version 10.00, make sure that the checksum routines are registered. On Informix version 10.00, checksum routines must be registered manually.
173	External Sync shutdown requested.	The synchronization task is not active. This error code is returned when Enterprise Replication is being shut down on a replication server participating in a synchronization task started by the **cdr sync** or **cdr check**command. **User action:** Run the **cdr list server** or **cdr view servers** command to see the status of the participating server and when all servers are active, rerun the original command.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
174	External Sync abort required.	The synchronization or repair task did not complete in the timeout period. This error can occur if the replicate being synchronized or the shadow replicate that was created to resynchronize the data is not active at all the participants specified in the command. This error code is also returned when the **cdr check replicate** or **cdr check replicateset** command is run with the **--enable** option and the target server cannot be enabled and repaired in the timeout period. The timeout period is 128 seconds or the value you set with the **--timeout** option. **User action:** Run the **cdr list replicate** command to check the replicate status. If all participants are active, try running the command again. If the server was being enabled, run the **cdr list server** command to check the server status. If all participants are active, try running the command again with an increased timeout value.
175	Sync has received a request to stop.	The synchronization or check command was stopped.
176	Sync attempted on replicate which is not active.	The synchronization or check command was stopped because one of the replicates specified is not active.
178	WARNING: Replicate is not in sync.	The replicate is not in sync. This error can be returned after running **cdr check replicate** or **cdr check replicateset**. This error can be returned after running **cdr check replicate** or **cdr check replicateset** with the **--repair** option if there are pending transactions that are not yet applied or if transactions were aborted. **User action:** If you receive this error after running a consistency check, repair the data. For more information, see Resynchronizing data among replication servers. If you receive this error code after repairing data, look for ATS or RIS files at target participants. If you see ATS or RIS files, look at the SQL and ISAM error code for the failures and if necessary repair the transactions by using the **cdr repair** command. If there are no ATS or RIS files at the target participants, rerun the original command with the **--inprogress** option to control how long check task rechecks inconsistent rows that might be in process of being applied at target servers.
181	Value specified cannot be set in-memory, for more information see message log file.	The specified configuration parameter was not modified for the current session. **User action:** For more information, see the server online message log file.
182	Warning: Value specified was adjusted before setting it in-memory, for more information see message log file.	The value of the configuration parameter specified in the command was adjusted and then the configuration parameter was reset for the current session. **User action:** For more information, see the server online message log file.
183	Operation not supported for the specified onconfig variable.	The specified configuration parameter cannot be dynamically updated while the server is running. **User action:** Edit the `onconfig` file and then shut down and restart the server.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
184	onconfig text is specified in wrong format.	The value specified for the configuration parameter is not valid. **User action:** For more information, see the server online message log file.
185	Specified variable is an unsupported or unknown ER onconfig or CDR_ENV variable.	The specified configuration parameter or environment variable is not valid in this command. **User action:** Check the spelling of the configuration parameter or environment variable.
186	Value of onconfig variable cannot be changed when ER is defined.	The specified configuration parameter cannot be changed after Enterprise Replication is initialized. **User action:** For more information, see the server online message log file.
187	Value of onconfig variable cannot be changed when HDR is defined.	The specified configuration parameter cannot be changed while the server is participating in a high-availability cluster.
188	WARNING: The onconfig variable is not modified as the specified value is same as stored in the memory.	The value specified for the configuration parameter is the same as its current value for the session.
189	Replicate cannot be defined or modified since the participant table is protected using Label Based Access Control.	The table specified in the command is using label-based access control (LBAC), which is not supported with Enterprise Replication. **User action:** Rerun the command with a different table name, or remove LBAC from the table and then rerun the command.
190	Code sets specified by CLIENT_LOCALE and DB_LOCALE must be identical.	The ATS or RIS file repair operation requires that the **CLIENT_LOCALE** and **DB_LOCALE** environment variables be set to the same value. **User action:** Reset the value of one of the environment variables to that it matches the other and then rerun the original command.
191	Cannot determine connection server ID for server.	The command cannot obtain the group ID for the server being connected to.
192	Unable to find or connect to a **syscdr** database at a non-leaf server.	The repair command canot find a root server from which to obtain the Enterprise Replication catalog information.
193	SQL failure due to server resource limitations.	An SQL statement failed with memory or lock resource-related error codes.
194	SQL failure due to loss of network connection to server.	An SQL query failed with a network error.
195	SQL failure.	This error code is returned when a command fails due to an SQL error code other than SQL resource limitations-related error codes.
196	Encountered an SQL error.	The command was stopped because an SQL statement failed.
200	Unexpected Internal Error with cdr check or cdr sync.	An internal UDR routine execution might have returned an unexpected error. **User action:** Look at the additional error messages printed on the screen to get more details about this error.
201	Sync/Check encountered an unexpected column type.	The data type of one of the columns being synchronized or checked cannot be resolved for data comparison.
202	Source and Target do not have the same data type.	Corresponding columns on the source server and the target server have different data types.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
203	Data for row or column not found.	Enterprise Replication cannot display the column value of a mismatched column on the screen.
204	Table could not be found.	The table that is being synchronized or repaired is not found on one of the participants. This error code is also returned if the participant being deleted cannot be found in the Enterprise Replication catalog tables.
205	Undefined server group.	The server group specified in the command was not found in the Enterprise Replication catalog tables. **User action:** Rerun the command with an existing server group name.
206	Template not realized at sync data source.	The template cannot be realized on the specified servers because it is not yet realized on the synchronization server. **User action:** Rerun the **cdr realize template** command specifying the synchronization source server as a participant.
207	Template already realized at one or more of requested servers.	One or more of the participants specified in the command already has the template instantiated on it. **User action:** Run the **cdr list replicate** command to check the status of the participants and then rerun the original command with the correct list of participants.
208	Server unknown at remote server.	Information about the local server is not available at the remote server.
209	A byte sequence that is not a valid character in the specified locale was encountered	One or more characters in a name specified in the command is not valid.
210	Parameter passed to command (or internally, routine) is invalid.	An argument specified in the command does not have a valid value.
211	Command is too large to be executed as a background task.	The command specified as a background task exceeded 2048 bytes. **User action:** Rerun the command without the **--background** option.
212	Sync/Check subtask aborted.	One of the tasks that was being performed in parallel was stopped. **User action:** Check the command output to determine which task was stopped.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
213	WARNING: set is not in sync.	At least one of the replicates in the specified replicate set is not in sync. This error can be returned after running **cdr check replicateset**. This error can be returned after running **cdr check replicateset** with the **--repair** option if there are pending transactions that are not yet applied or if transactions were aborted. **User action:** If you receive this error after running a consistency check, repair the data. For more information, see Resynchronizing data among replication servers. If you receive this error code after repairing data, look for ATS or RIS files at target participants. If you see ATS or RIS files, look at the SQL and ISAM error code for the failures and if necessary repair the transactions by using the **cdr repair** command. If there are no ATS or RIS files at the target participants, rerun the original command with the **--inprogress** option to control how long the check task rechecks inconsistent rows that might be in process of being applied at target servers.
214	ER: The logical log replay position is not valid. Restart ER with the **cdr cleanstart** command, and then synchronize the data with the **cdr check --repair** command.	Enterprise Replication cannot start because the logical log replay position is not valid. **User action:** Run the **cdr cleanstart** command and then the **cdr check replicateset** command with the **--repair** option.
215	Command failed -- The specified table is an external table. You cannot include an external table in a replicate.	Tables created with the CREATE EXTERNAL TABLE statement cannot be included in a replicate.
217	Error with Quality of Data command.	This error can be returned after running the **cdr define qod** command if the quality of data master server is already defined. This error can be returned after running the **cdr start qod** command or the **cdr stop qod** command if the quality of data master server is not defined or the command was run from a different server. **User action:** If the quality of data master does not exist, run the **cdr define qod** command and then rerun the original command. If the command was run on a different server, rerun the command from the quality of data master server, as indicated in the error message.
220	A node included in the list is not valid.	The server group specified in the grid command was not found in the Enterprise Replication catalog tables. **User action:** Rerun the command with an existing server group name.
221	The grid name is not unique.	Grid names must be unique among grids and among replicate sets. **User action:** Run the **cdr list grid** command to see existing grid names and run the **cdr list replicateset** command to see existing replicate set names. Rerun the original command with a unique grid name.
222	The grid does not exist.	The grid name specified in the command is not the name of an existing grid. **User action:** Run the **cdr list grid** command to see existing grid names. Rerun the original command with an existing grid name.

Table A-1. Return codes for the cdr utility (continued)

Return code	Error text	Explanation
223	grid enable user failed	The user name specified in the command does not exist. **User action:** Rerun the command with an existing user name.
224	grid enable node failed	The server name specified in the command is not the name of an existing replication server. **User action:** Rerun the command with an existing replication server name.
225	sec2er failure	The **cdr start sec2er** command failed. **User action:** Following the instructions in the command output to perform all necessary prerequisites.

Related reference:

"cdr finderr" on page A-93

Frequency Options

You can specify the interval between replications or the time of day when replication should occur for a replicate.

The frequency of replication is a property of a replicate. You can set the frequency of replication for a replicate when you define it or modify it. You can reset the frequency of all replicates in a replicate set when you define or modify a replicate set or define a template. For non-exclusive replicate sets, you can update the frequency of individual replicates separately.

If you do not specify a time, replication occurs immediately.

Frequency Options:

Element	Purpose	Restrictions
interval	Time interval for replication	The smallest interval in minutes, in one of the following formats: • The number of minutes, as an integer value between 1 and 1966020, inclusive. • The number of hours and minutes separated by a colon. The minimum value is 0:01. The maximum value is 32767:59
time	Specific time for replication	Time is given with respect to a 24-hour clock.

The following table describes the frequency 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.

Intervals

The **-e** *interval* option lets you specify the interval between actions. The *interval* 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 greater than 0. For example, `-e 60` indicates 60 minutes between replications.

  If you specify the interval 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.

If you specify the length of time in hours and minutes, the longest interval allowed is 32767:59.

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, `--at=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 `--at=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 day 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 A.M.

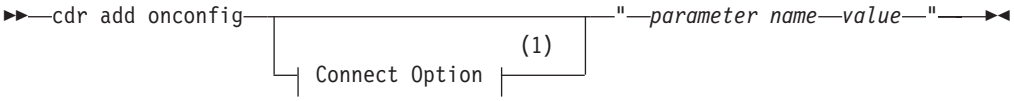
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicate" on page A-61
"cdr define replicateset" on page A-69
"cdr define template" on page A-75
"cdr modify replicate" on page A-108
"cdr modify replicateset" on page A-111

cdr add onconfig

The **cdr add onconfig** command adds one or more values to a configuration parameter in the ONCONFIG file.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter to set.	You can add values to the following Enterprise Replication configuration parameters: • CDR_LOG_LAG_ACTION • CDR_LOG_STAGING_MAXSIZE • CDR_QDATA_SBSPACE • CDR_SUPPRESS_ATSRISWARN • ENCRYPT_MAC • ENCRYPT_MACFILE • CDR_ENV: – CDRSITES_731 – CDRSITES_92X – CDRSITES_10X	
value	The value of the configuration parameter.	Must be a valid value for the configuration parameter.	Follows the syntax rules for the specific configuration parameter.

Usage

Use the **cdr add onconfig** command to add one or more values to an Enterprise Replication configuration parameter while replication is active. The ONCONFIG file is updated. You can set environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example adds an sbspace to the existing list of sbspaces for holding spooled transaction row data:

```
cdr add onconfig "CDR_QDATA_SBSPACE sbspace_11"
```

The following example adds the cdrIDs for two version 7.x servers to the existing list of servers:

```
cdr add onconfig "CDR_ENV CDRSITES_731=1,3"
```

Related tasks:

"Dynamically Modifying Configuration Parameters for a Replication Server" on page 8-1

Related reference:

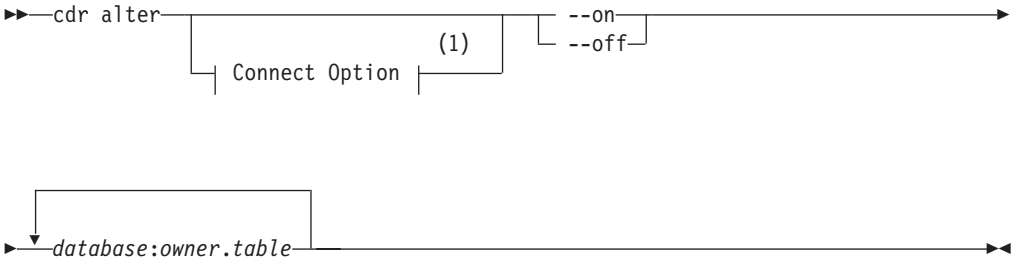
"cdr change onconfig" on page A-31

"cdr remove onconfig" on page A-121

cdr alter

The **cdr alter** command puts the specified tables in alter mode.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
database	The name of the database that contains the table	The database server must be registered with Enterprise Replications.	"Long Identifiers" on page A-3
owner	User ID of the owner of the table		"Long Identifiers" on page A-3
table	Specifies the name of the table to put in alter mode	The table must be an actual table. It cannot be a synonym or a view.	"Long Identifiers" on page A-3

The following table describes the options to **cdr alter**.

Long Form	Short Form	Meaning
--on	**-o**	Sets alter mode on.
--off	**-f**	Unsets alter mode.

Usage

Use this command to place a table in or out of alter mode. Use alter mode when you need to alter an attached fragment to the table or you want to perform other alter operations manually.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example puts **table1** and **table2** in alter mode:

```
cdr alter --on db1:owner1.table1 db2:owner2.table2
```

Related concepts:

"Alter, Rename, or Truncate Operations during Replication" on page 8-22

"Limited SQL Statements" on page 2-11

Related reference:

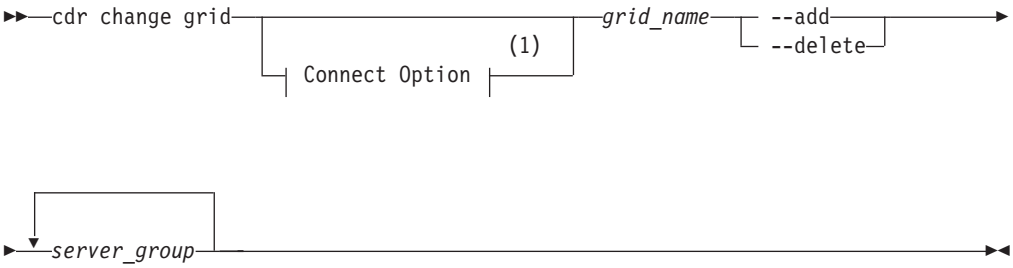
"cdr swap shadow" on page A-159

"cdr remaster" on page A-119

cdr change grid

The **cdr change grid** command adds or deletes replication servers to or from a grid.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	"Long Identifiers" on page A-3
server_group	Name of a database server group to add to, or remove from, the grid.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3

The following table describes the **cdr change grid** options.

Long Form	Short Form	Meaning
--add	**-a**	Add the specified replication servers to the grid.

Long Form	Short Form	Meaning
--delete	**-d**	Delete the specified replication servers from the grid.

Usage

Use the **cdr change grid** command to add a new replication server to an existing grid or to remove a replication server from an existing grid.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see "Return Codes for the cdr Utility" on page A-8.

Examples

The following example adds two servers to a grid named **grid_1**:

```
cdr change grid grid_1 --add gserv3 gserv4
```

The following example removes a server from a grid named **grid_1**:

```
cdr change grid grid_1 --delete gserv1
```

Related tasks:

"Maintaining the grid" on page 7-5

"Adding a server to a grid by cloning" on page 7-5

Related reference:

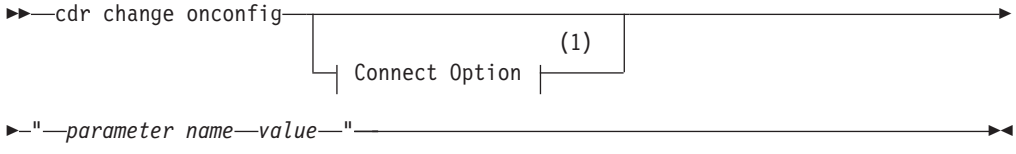
"cdr define grid" on page A-58

"cdr list grid" on page A-94

cdr change onconfig

The **cdr change onconfig** command replaces the existing value of a configuration parameter with a new value in the ONCONFIG file.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter to change.	None. All Enterprise Replication configuration parameters and environment variables can be changed with this command.	
value	The value of the configuration parameter.	Must be a valid value for the configuration parameter.	Follows syntax rules for the specific configuration parameter.

Usage

Use the **cdr change onconfig** command to replace the existing value of an Enterprise Replication configuration parameter with a new value in the ONCONFIG file. You can set Enterprise Replication environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

Suppose the CDR_SUPPRESS_ATSRISWARN configuration parameter is set to suppress the generation of error and warning messages 1, 2, and 10, so that it appears in the ONCONFIG file as: CDR_SUPPRESS_ATSRISWARN 1,2,10. The following command changes the suppressed error and warning messages to 2, 3, 4, 5, and 7:

```
cdr change onconfig "CDR_SUPPRESS_ATSRISWARN 2-5,7"
```

Suppose the CDR_RMSCALEFACT environment variable is set to the value of 4. The following example sets the number of data sync threads started for each CPU VP to 3:

```
cdr change onconfig "CDR_ENV CDR_RMSCALEFACT=3"
```

Related tasks:

"Dynamically Modifying Configuration Parameters for a Replication Server" on page 8-1

Related reference:

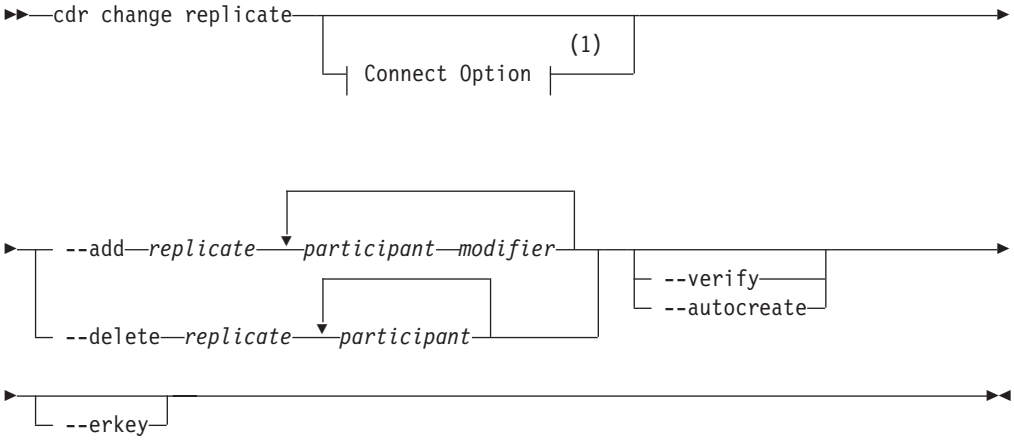
"cdr add onconfig" on page A-28

"cdr remove onconfig" on page A-121

cdr change replicate

The **cdr change replicate** command modifies an existing replicate by adding or deleting one or more participants.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		"Participant and participant modifier" on page A-4
participant	Specifies the database server and table for replication.	The participant must exist.	"Participant and participant modifier" on page A-4
replicate	Name of the replicate to change.	The replicate must exist.	"Long Identifiers" on page A-3

The following table describes the options to **cdr change replicate**.

Long Form	Short Form	Meaning
--add	**-a**	Adds participants to a replicate.
--autocreate	**-u**	For use with master replicates only. Specifies that if the tables in the master replicate definition do not exist in the databases on the target servers, then they are created automatically. However, the table cannot contain columns with user-defined data types. The tables are created in the same dbspace as the database. **Note:** Tables created with the **--autocreate** option do not automatically include non-primary key indices, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with the **--autocreate** option require the use of these objects you must explicitly create the objects by hand.
--delete	**-d**	Removes participants from a replicate.

Long Form	Short Form	Meaning
--erkey	**-K**	Includes the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, in the replicate definition, if the table being replicated has the ERKEY shadow columns. The ERKEY shadow columns are used in place of a primary key.
--verify	**-v**	For use with master replicates only. Specifies that the **cdr change template** command verifies the database, tables, and column data types against the master replicate definition on all listed servers

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. 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-131.

When you run the **cdr change replicate** command, an event alarm with a class ID of 65 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

Example 1: Add two participants

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"
```

Example 2: Remove two participants

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"
```

Example 4: Add a participant that includes ERKEY shadow columns

The following example adds a participant and includes the ERKEY shadow columns from the table **table1**:

```
cdr change repl -a repl_1 --erkey\
   "db1@server1:antonio.table1" "select * from table1"
```

Related tasks:

"Preparing tables without primary keys" on page 4-21

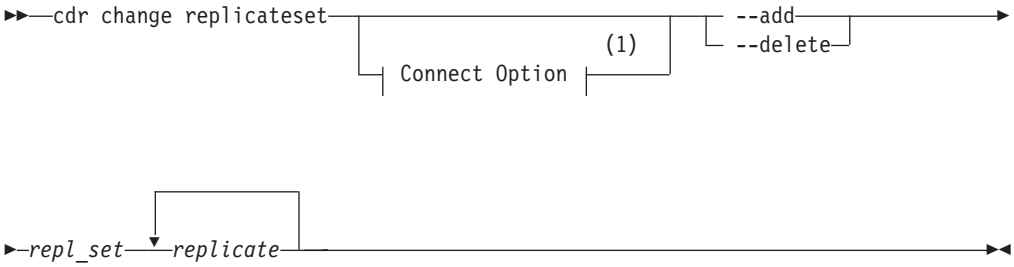
Related reference:

"cdr define replicate" on page A-61
"cdr delete replicate" on page A-79
"cdr list replicate" on page A-97
"cdr modify replicate" on page A-108
"cdr resume replicate" on page A-126
"cdr start replicate" on page A-131
"cdr stop replicate" on page A-152
"cdr suspend replicate" on page A-155
"Enterprise Replication Event Alarms" on page 9-21
"Participant and participant modifier" on page A-4

cdr change replicateset

The **cdr change replicateset** command changes a replicate set by adding or deleting replicates.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set to change.	The replicate set must exist.	"Long Identifiers" on page A-3
replicate	Name of the replicates to add to or delete from the set.	The replicates must exist.	"Long Identifiers" on page A-3

The following table describes the options to **cdr change replicateset**

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 or that is defined with a frequency interval, 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.

Use this command to add or remove replicates from a grid replicate set. You can only add replicates that were created outside of a grid environment to a grid replicate set if the following conditions are met:

- The participant servers must be the same as the servers in the grid.
- The replicated table schema must be the same among all participants.
- The entire replicated table is replicated. Using a SELECT statement in the participant definition that does not include all the columns in the table or includes a WHERE clause is not allowed.

When you run the **cdr change replicateset** command, an event alarm with a class ID of 66 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

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 change replset --delete favorite_set teepee wigwam
```

Related concepts:

"Frequency Options" on page A-26

Related tasks:

"Suspending a Replicate Set" on page 8-12

"Adding an existing replicate to a grid replicate set" on page 7-7

Related reference:

"cdr define replicate" on page A-61

"cdr delete replicateset" on page A-80

"cdr list replicateset" on page A-101

"cdr modify replicateset" on page A-111

"cdr resume replicateset" on page A-127

"cdr start replicateset" on page A-134

"cdr stop replicateset" on page A-154

"cdr suspend replicateset" on page A-157

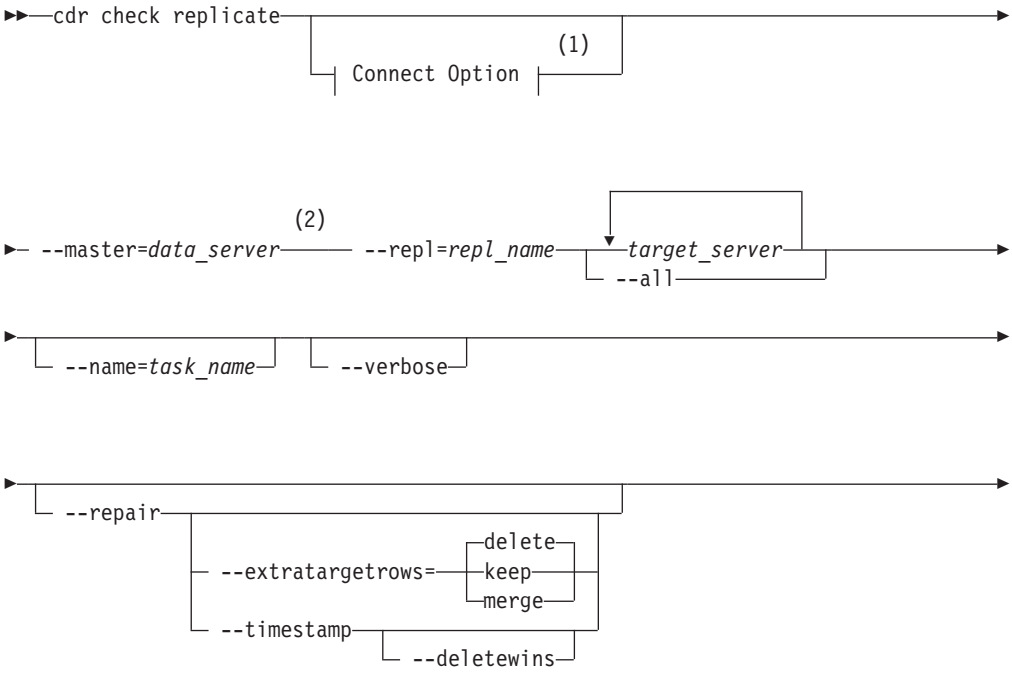
"cdr define replicateset" on page A-69

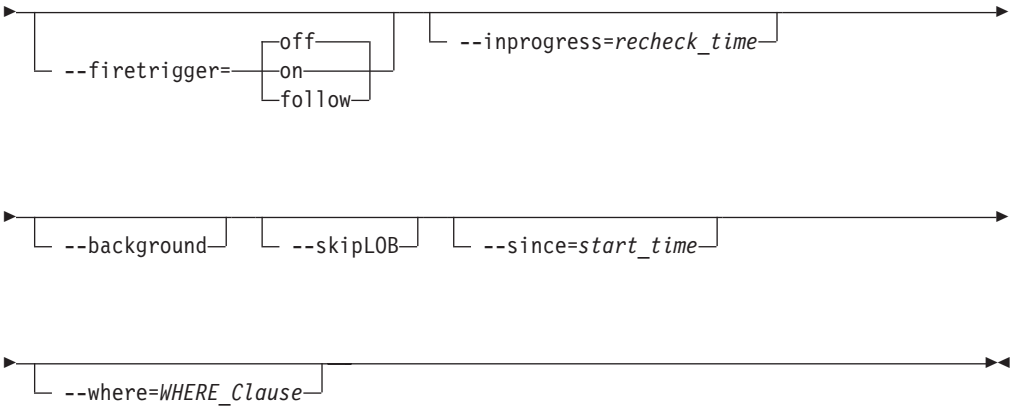
"Enterprise Replication Event Alarms" on page 9-21

cdr check replicate

The **cdr check replicate** command compares the data on replication servers to create a report listing data inconsistencies and can optionally repair the inconsistent data within a replicate.

Syntax





Notes:

1 See "Connect Option" on page A-3.

2 Omit if you include the **--timestamp** option.

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
recheck_time	The number of seconds to spend rechecking transactions that might be listed as inconsistent because they are not yet applied on the target server.	Must be a positive integer.	
repl_name	Name of the replicate to check.	Must be an existing replicate.	"Long Identifiers" on page A-3

Element	Purpose	Restrictions	Syntax
start_time	The time from which to check updated rows.	Can have one the following formats: • *number***M** = Include rows updated in the last specified number of minutes. • *number***H** = Include rows updated in the last specified number of hours. • *number***D** = Include rows updated in the last specified number of days. • *number***W** = Include rows updated in the last specified number of weeks. • "*YYYY-MM-DD hh*:*mm*:*ss*" = Include rows updated since this time stamp.	The time stamp format follows the convention of the DBTIME environment variable.
target_server	Name of a database server group to check.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	"Long Identifiers" on page A-3
WHERE_Clause	Clause that specifies a subset of table rows to be checked.		WHERE clause syntax

The following table describes the **cdr check replicate** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are checked.
--background	**-B**	Specifies that the operation is performed in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name specified with the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--deletewins	**-d**	Specifies that the replicate uses the delete wins conflict resolution rule.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while synchronizing the data: • **off**: (default) do not fire triggers at target servers during synchronization • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option
--inprogress=	**-i**	Specifies to spend more than the default time rechecking inconsistent rows that might be in the process of being applied on target servers. If the **--inprogress=** option is not set, inconsistent rows are rechecked for up to five seconds.
--master=	**-m**	Specifies the database server to use as the reference copy of the data. You cannot use the **--master** option with the **--timestamp** option.
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--repair	**-R**	Specifies that rows that are found to be inconsistent are repaired.
--repl=	**-r**	Specifies the name of the replicate to check.
--since=	**-S**	Specifies the time from which to check updated rows. The replicate must be using the time stamp or delete wins conflict resolution rule.
--skipLOB	**-L**	Specifies that large objects are not checked.
--timestamp	**-t**	Specifies to repair inconsistent rows based on the latest time stamp among all the participants. The replicate must use the time stamp or delete wins conflict resolution rule. You cannot use the **--master** option with the **--timestamp** option.
--verbose	**-v**	Specifies that the consistency report shows specific inconsistent values.
--where=	**-w**	Specifies what data to check with a WHERE clause.

Usage

Use the **cdr check replicate** command to check the consistency of data between multiple database servers for a specific replicate. The **cdr check replicate** command compares all rows on all specified database servers against the data in the reference server and produces a consistency report. If you include the **--verbose** option, the report lists every inconsistent row value; otherwise, the report summarizes inconsistent rows.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

If you want to monitor the progress of the check operation, include the **--name** option and specify a name for the progress report task. Then run the **cdr stats check** command and specify the progress report task name.

If replicated transactions are active when the **cdr check replicate** command is running, the consistency report might include rows that are temporarily inconsistent until those transactions are applied at the target server. By default, the **cdr check replicate** command rechecks inconsistent rows for up to five seconds after the initial check is completed. If you find your transaction latency is longer than five seconds, you can extend the recheck time period by using the **--inprogress** option to specify a longer interval. After the initial recheck, inconsistent transactions are rechecked until there are no inconsistent transactions or the number of seconds specified by the **--inprogress** option elapses. In general, set the recheck time to a little longer than your average transaction latency because if repairing inconsistencies causes spooling in the send queue, transaction latency might increase during a repair. View your transaction latency with the **cdr view apply** command, or in the IBM OpenAdmin Tool (OAT) for Informix.

You can improve the performance of consistency checks by limiting the amount of data that is checked by using one or more of the following options:

- Check from a specific time with the **--since** option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.
- Check a subset of the data with the **--where** option. For example, if you have a corrupted table fragment on a server, you can specify to check only the data in that fragment.
- Skip the checking of large objects with the **--skipLOB** option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.

You can run a consistency check as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the **--name** option so that you can monitor the check with the **cdr stats check** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database. If you use the **--background** option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If you have large tables, you can speed consistency checking by indexing the **ifx_replcheck** shadow column.

If you include the **--repair** option, the **cdr check replicate** command repairs inconsistent rows so that they match the rows on the reference server. The **cdr check replicate** command uses direct synchronization as a foreground process when repairing inconsistent rows. The **cdr check replicate** command with the **--repair** option performs the following tasks:

1. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
2. Performs an index scan using the primary key index at both the source server and the target server to create a checksum and identify inconsistent rows.
3. Replicates inconsistent rows from the source server to the target server by performing a dummy update of the source server, which might result in increase logging activity.
4. Runs a check to determine if any rows remain inconsistent. Rows can be temporarily inconsistent if not all transactions are complete on the target server.
5. If any rows are inconsistent, reruns the check for up to five seconds, or for up to the number of seconds specified by the **--inprogress** option.
6. Deletes the shadow replicate.
7. Displays the consistency report.

To repair replicate sets based on the latest time stamps among the participants instead of based on a master server, use the **--repair** option with the **--timestamp** option. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option. A time stamp repair evaluates extra and mismatched rows according to the rules of the time stamp or delete wins conflict resolution rules. The reference server in a time stamp repair is the server with the lowest primary key.

The following table describes the columns of the consistency report.

Table A-2. Consistency Report Description

Column name	Description
Node	The name of the replication server.
Rows	The number of rows checked in the participant. If you included the **--since** or **--where** options, this number indicates the number of rows that fit the filter conditions. The number of rows checked with the **--since** option might be different on different servers, because of replication latency. Some rows might be checked on a source server to verify target server rows even if those rows on the source server did not originally fit the filter conditions.
Extra	The number of rows on the target server that do not exist on the reference server. For the reference server, this number is always 0.
Missing	The number of rows on the reference server that do not exist on the target server. For the reference server, this number is always 0.

Table A-2. Consistency Report Description (continued)

Column name	Description
Mismatch	The number of rows on the target server that are not consistent with the corresponding rows on the reference server. For the reference server, this number is always 0.
Processed	The number of rows processed to correct inconsistent rows. The number of processed rows on the reference server is equal to the number of mismatched rows plus missing rows on the target servers. The number of processed rows for a target server is usually equal to the number of extra rows it has. If a row has child rows, then the number of processed rows can be greater than the number of extra rows because the child rows must be deleted as well. If the **--extratargetrows** option is set to **keep**, then extra rows are not deleted from the target and those rows are not added to the Processed column. If the **--extratargetrows** option is set to **merge**, then those rows are replicated to the reference server and are listed in the Processed column for the target server. For a time stamp repair, the time stamp or delete wins conflict resolution rule is used to determine how to process the row.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 17, 18, 31, 37, 48, 53, 54, 61, 75, 99, 101, 121, 172, 174, 178, 193, 194, 195, 200, 203, 204.

For information on these error codes, see "Return Codes for the cdr Utility" on page A-8

Examples

These examples show some of the options available for checking consistency.

Example 1: Summary consistency report

The following command generates a consistency report for a replicate named **repl1**, comparing the data on the server **serv2** with the data on the server **serv1**:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2
```

The summary consistency report for the previous command might be:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 52         0         0         0         0
```

```
g_serv2                 52         0         0         0         0

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

This report indicates that the replicate is consistent on these servers.

Example 2: Summary consistency report with repair

The following command generates a consistency report and repairs inconsistent rows on all servers for a replicate named **repl1**:

```
cdr check replicate --master g_serv1 --repl=repl1 --all --repair
```

If a target server has extra rows, the consistency report for the previous command might be:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         2
g_serv2                 67         2         2         0         2
g_serv3                 67         0         0         0         0

Validation of repaired rows failed.
WARNING: replicate is not in sync

Jan 17 2009 15:46:55 ------   Table scan for repl1 end   ---------
```

This report indicates that **g_serv2** has two extra rows and is missing two rows. Two rows were processed on **g_serv1** to replicate the missing rows to **g_serv2**. Also, two rows were processed on **g_serv2** to delete the extra rows. Because the **--extratargetrows** option was not specified, the default behavior of deleting rows on the target servers that are not on the reference server occurred.

In this example, not all repaired rows were validated. Some rows might be still in the process of being applied on the target servers. Using the **--inprogress** option to extend the time of the validation check after the repair might prevent validation failures.

Example 3: Verbose consistency report with repair

The following command generates a verbose consistency report, creates a progress report task, and repairs inconsistent rows on all servers for a replicate named **repl1**:

```
cdr check replicate --master=g_srv1 --replicate=repl1 --all --name=task1 \
 --verbose --repair
```

The verbose consistency report for the previous command might be:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Creating Shadow Repl sync_20104_1310721_1219952381
Node                  Rows     Extra   Missing  Mismatch Processed
----------------- --------- --------- --------- --------- ---------
g_srv1                 424         0         0         0        11
g_srv2                 416         3        11         0         3

The repair operation completed. Validating the repaired rows ...
Validation failed for the following rows:
```

```
row missing on <g_srv2>
key:c1:424
------------------------------------------------------------
row missing on <g_srv2>
key:c1:425
------------------------------------------------------------
row missing on <g_srv2>
key:c1:426
------------------------------------------------------------
marking completed on g_srv1 status 0

Jan 17 2009 15:46:59 ------   Table scan for repl1 end   ---------
```

This report indicates that the first check found three extra rows and 11 missing rows on the server **g_srv2**. After the repair operation and subsequent recheck, three rows were still missing on **g_srv2**. The progress report information can be accessed with the **cdr stats check task1** command.

Example 4: Repeating verbose consistency report without repair

The following command generates a verbose consistency report for a replicate named **repl1**, comparing the data on the server **serv2** with the data on the server **serv1**, and rechecks inconsistent rows for up to 20 seconds:

```
cdr check replicate --master g_serv1 --repl=repl_1 g_serv2 --all \
--verbose --inprogress=20
```

The verbose consistency report for the previous command might be:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
data mismatch between g_serv1 and g_serv2
item_num:1
order_num:1011
          lname
g_serv1   Pauly
g_serv2   Pauli
-------------------------------------------------
row missing on g_serv2
item_num:1
order_num:1014
-------------------------------------------------
row missing on g_serv2
item_num:2
order_num:1014
-------------------------------------------------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         0         0
g_serv2                 65         0         2         1         0

WARNING: replicate is not in sync

Jan 17 2009 15:47:15 ------   Table scan for repl1 end   ---------
```

This report indicates that there is one inconsistent row on **g_serv2**. The primary key for that row is the combination of the **item_num** column and the **order_num** column. The row that is inconsistent is the one that has the item number 1 and the order number 1011. There are two rows that are missing on **g_serv2**, each identified by its primary key value.

Example 5: Summary consistency report with time filter

The following command generates a summary consistency report for the data that was updated in the last five minutes:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2 --since=5M
```

The consistency report for the previous command might be:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                  2         0         0         0         0
g_serv2                  2         0         0         0         0

Jan 17 2009 15:46:50 ------   Table scan for repl1 end   ---------
```

Only two rows were checked on each server (the Rows column) because only two rows were updated in the last five minutes.

Example 6: Consistency check and repair with time filter

The following command generates a summary consistency report for the data that was updated since July 4, 2008 at 12:30:00 local time:

```
cdr check replicate --master=g_serv1 --repl=repl1 g_serv2 \
--since="2008-07-04 12:30:00"
```

Example 7: Summary consistency report and repair with data filters

The following command generates a consistency report and repairs the data where the **region** column equals `East`:

```
cdr check replicate --master=g_serv1 --repl=repl1 --repair g_serv2 \
--where="region = 'East'"
```

Example 8: Repair inconsistencies based on time stamp

The following command repairs inconsistencies based on the most recent time stamps for the **repl1** replicate on all replication servers:

```
cdr check replicate --repl=repl1 --all --repair --timestamp
```

The master server is not specified because the **--timestamp** option is used.

The consistency report for the previous command might be:

```
Jan 17 2009 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 67         0         0         4        10
g_serv2                 67         0         2         3         0
g_serv3                 67         0         5         0         4

WARNING: replicate is not in sync

Jan 17 2009 15:46:55 ------   Table scan for repl1 end   ---------
```

The value in the Extra column is always 0. In this example, seven rows are replicated from the **g_serv1** server to fix missing rows. The **g_serv1** server also replicated three rows to fix mismatched rows on the **g_serv2** server. The **g_serv3** server replicated four rows to resolve mismatched rows on the **g_serv1** server.

Related concepts:

"Interpreting the Consistency Report" on page 8-18
"Database Server Groups" on page 4-2
"Preparing for Role Separation (UNIX)" on page 4-22

Related tasks:

"Checking Consistency and Repairing Inconsistent Rows" on page 8-17
"Indexing the ifx_replcheck Column" on page 8-19
"Increase the speed of consistency checking" on page 8-19

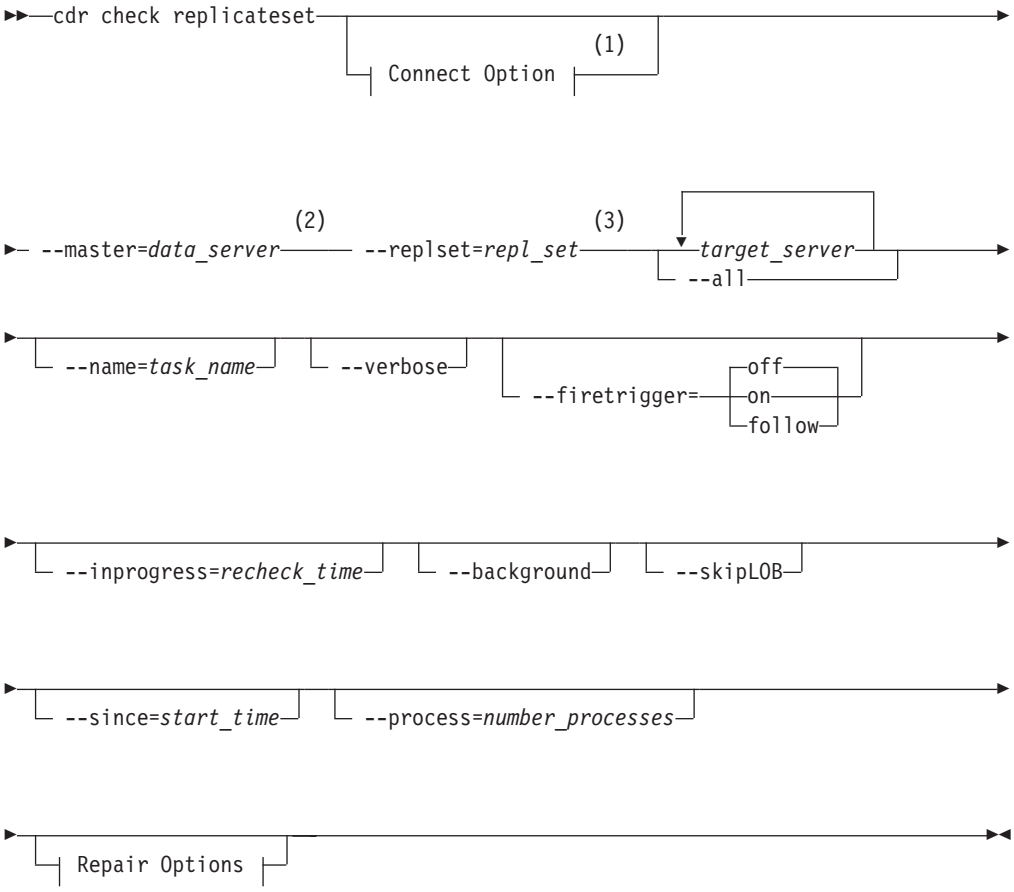
Related reference:

"cdr sync replicate" on page A-161
"cdr check replicateset"
"cdr stats check" on page A-143

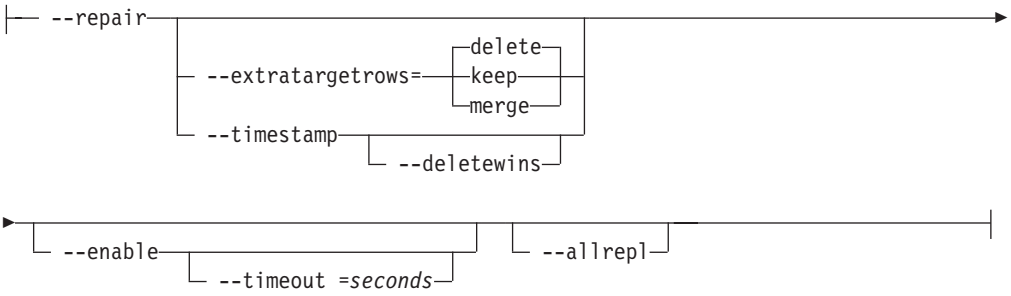
cdr check replicateset

The **cdr check replicateset** command compares the data on replication servers to create a report listing data inconsistencies. Optionally you can use the command to repair the inconsistent data within a replicate.

Syntax



Repair Options:



Notes:

1 See "Connect Option" on page A-3.

2 Omit if you include the **--timestamp** option.

3 Omit if you include the **--allrepl** option.

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
number_processes	The number of parallel processes to use for the command.	The maximum number of processes Enterprise Replication can use is equal to the number of replicates in the replicate set.	
recheck_time	The number of seconds to spend rechecking transactions that might be listed as inconsistent because they are not yet applied on the target server.	Must be a positive integer.	
repl_set	Name of the replicate set to synchronize.		"Long Identifiers" on page A-3
seconds	The number of seconds to wait for a disabled replication server to be recognized as active by other replication servers in the domain and how long to wait for control messages queued at peer servers to be applied at newly-enabled server.	Must be an integer value from 0 to 60.	

Element	Purpose	Restrictions	Syntax
start_time	The time from which to check updated rows.	Can have one the following formats: • *number***M** = Include rows updated in the last specified number of minutes. • *number***H** = Include rows updated in the last specified number of hours. • *number***D** = Include rows updated in the last specified number of days. • *number***W** = Include rows updated in the last specified number of weeks. • "*YYYY-MM-DD hh*:*mm*:*ss*" = Include rows updated since this time stamp.	The time stamp format follows the convention of the DBTIME environment variable.
target_server	Name of a database server group to check.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	"Long Identifiers" on page A-3

The following table describes the **cdr check replicateset** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are checked.
--allrepl	**-A**	Specifies that all replicates, whether they are in a replicate set or not, are repaired. You cannot use the **--replset** option with the **--allrepl** option.
--background	**-B**	Specifies that the operation is performed in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name specified with the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--enable	**-E**	Enables replication on the target server if it was disabled by the **cdr disable server** command.
--deletewins	**-d**	Specifies to use the delete wins conflict resolution rule when performing a time stamp repair.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while synchronizing the data: • **off**: (default) do not fire triggers at target servers during synchronization • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option
--inprogress=	**-i**	Specifies to spend more than the default time rechecking inconsistent rows that might be in the process of being applied on target servers. If the **--inprogress=** option is not set, inconsistent rows are rechecked for up to five seconds.
--master	**-m**	Specifies the database server to use as the reference copy of the data. You cannot use the **--master** option with the **--timestamp** option.
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--process=	**-p**	Specifies to run the command in parallel, using the specified number of processes. At most, Enterprise Replication can use one process for each replicate in the replicate set. If you specify more processes than replicates, the extra processes are not used. Not all replicates can be processed in parallel. For example, if replicates have referential integrity rules, the replicates with the parent tables must be processed before the replicates with the child tables.
--repair	**-R**	Specifies that rows that are found to be inconsistent are repaired.
--replset	**-s**	Specifies the name of the replicate set to check. You cannot use the **--replset** option with the **--allrepl** option.
--skipLOB	**-L**	Specifies that large objects are not checked.
--since=	**-S**	Specifies the time from which to check updated rows. The replicate must be using the time stamp or delete wins conflict resolution rule.
--timeout=	-w	

Long Form	Short Form	Meaning
--timestamp	**-t**	Specifies to repair inconsistent rows based on the latest time stamp among all the participants. All replicates in the replicate set must use the time stamp or delete wins conflict resolution rule. You cannot use the **--master** option with the **--timestamp** option.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr check replicateset** command to check the consistency of data between multiple database servers for a replicate set. The **cdr check replicateset** command compares all rows on all specified database servers against the data in the reference server and produces a consistency report. If you include the **--verbose** option, the report lists every inconsistent value; otherwise, the report summarizes inconsistent rows.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

If you want to monitor the progress of the check operation, include the **--name** option and specify a name for the progress report task. Then run the **cdr stats check** command and specify the progress report task name.

If replicated transactions are active when the **cdr check replicateset** command is running, the consistency report might include rows that are temporarily inconsistent until those transactions are applied at the target server. By default, the **cdr check replicateset** command rechecks inconsistent rows for up to five seconds after the initial check is completed. If you find your transaction latency is longer than five seconds, you can extend the recheck time period by using the **--inprogress** option to specify a longer interval. After the initial recheck, inconsistent transactions are rechecked until there are no inconsistent transactions or the number of seconds specified by the **--inprogress** option elapses. In general, set the recheck time to a little longer than your average transaction latency because if repairing inconsistencies causes spooling in the send queue, transaction latency might increase during a repair. View your transaction latency with the **cdr view apply** command, or in the IBM OpenAdmin Tool (OAT) for Informix.

You can improve the performance of consistency checks by limiting the amount of data that is checked by using one or more of the following options:

- Skip the checking of large objects with the **--skipLOB** option. If you find that your large objects do not change as much as other types of data, then skipping them can make a consistency check quicker.
- Check from a specific time with the **--since** option. If the replicate uses the time stamp or delete wins conflict resolution rule and you regularly check consistency, you can limit the data that is checked to the data that was updated since the last consistency check.

You can significantly improve the performance of checking a replicate set by checking the member replicates in parallel. You specify the number of parallel processes with the **--process** option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

You can run a consistency check as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular consistency checks with the Scheduler. If you run a consistency check in the background, provide a name for the progress report task by using the **--name** option so that you can monitor the check with the **cdr stats check** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database. If you use the **--background** option as a DBSA, you must have CONNECT privilege on the **sysadmin** database and INSERT privilege on the **ph_task** table.

If you have large tables, you can speed consistency checking by indexing the **ifx_replcheck** shadow column.

The **cdr check replicateset** command repairs inconsistent rows so that they match the rows on the reference server. During a repair of inconsistent rows, the **cdr check replicateset** command uses direct synchronization as a foreground process when repairing inconsistent rows. The **cdr check replicateset** command with the **--repair** option performs the following tasks:

1. Determines the order in which to repair tables if they have referential relationships.
2. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
3. Performs an index scan using the primary key index at both the source server and the target server to create a checksum and identify inconsistent rows.
4. Replicates inconsistent rows from the source server to the target server by performing a dummy update of the source server, which might result in increase logging activity.
5. Runs a check to determine if any rows remain inconsistent. Rows can be temporarily inconsistent if not all transactions are complete on the target server.
6. If any rows are inconsistent, reruns the check for up to five seconds, or for up to the number of seconds specified by the **--inprogress** option.
7. Deletes the shadow replicate.
8. Repeats steps 2 through 7 for each replicate in the replicate set.
9. Displays the consistency report.

If you have disabled a server with the **cdr disable server** command, you can enable it and synchronize it by using the **--enable** option with the **--repair** option. You can optionally specify a timeout period with the **--timeout** option.

To repair all replicates, use the **--allrepl** option with the **--repair** option.

To repair replicate sets based on the latest time stamps among the participants instead of based on a master server, use the **--repair** option with the **--timestamp** option. If your replicates use the delete wins conflict resolution rule, also include the **--deletewins** option. A time stamp repair evaluates extra and mismatched rows according to the rules of the time stamp or delete wins conflict resolution rules.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 11, 17, 18, 31, 37, 48, 53, 54, 61, 75, 99, 101, 121, 166, 172, 174, 193, 194, 195, 200, 203, 204, 213.

For information about these error codes, see "Return Codes for the cdr Utility" on page A-8

Examples

The following examples show how to use the **cdr check replicateset** command.

Example 1: Generate a consistency report

The following command uses two processes to generate a consistency report for each of the two replicates in the set in parallel for a replicate set named **replset1**, comparing the data on the server **serv2** with the data on the server **serv1**:

```
cdr check replicateset --master=g_serv1 --replset=replset_1 g_serv2 \
--process=2
```

The summary consistency report for the previous command might be:

```
Jan 17 2010 15:46:45 ------   Table scan for repl1 start  --------

------   Statistics for repl1 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 52         0         0         0         0
g_serv2                 52         0         0         0         0

Jan 17 2010 15:46:55 ------   Table scan for repl1 end   ---------

Jan 17 2010 15:46:46 ------   Table scan for repl2 start  --------

------   Statistics for repl2 ------
Node                  Rows     Extra   Missing  Mismatch Processed
--------------- --------- --------- --------- --------- ---------
g_serv1                 48         0         0         0         0
g_serv2                 48         0         0         0         0

Jan 17 2010 15:47:05 ------   Table scan for repl2 end   ---------
```

This report indicates that the replicate set is consistent on these servers.

The consistency report for replicate sets shows a series of consistency reports for individual replicates that has the same format as the reports run with the **cdr check replicate** command.

Example 2: Enable and synchronize a replication server

The following command enables a replication server named **g_serv2** and repairs inconsistencies by time stamp on all of its replicate sets:

```
cdr check replicateset  --repair --enable\
--timestamp --allrepl g_serv2
```

The master server is not specified because the **--timestamp** option is used. The replicate set name is not specified because the **--allrepl** option is used.

Example 3: Repair inconsistencies based on time stamp

The following command repairs inconsistencies based on the most recent time stamps for all replicate on all replication servers:

```
cdr check replicateset --all --repair --timestamp --allrepl
```

Related concepts:

"Database Server Groups" on page 4-2

"Preparing for Role Separation (UNIX)" on page 4-22

Related tasks:

"Checking Consistency and Repairing Inconsistent Rows" on page 8-17

"Indexing the ifx_replcheck Column" on page 8-19

"Increase the speed of consistency checking" on page 8-19

"Repairing inconsistencies while enabling a replication server" on page 8-21

Related reference:

"cdr sync replicateset" on page A-164

"cdr check replicate" on page A-37

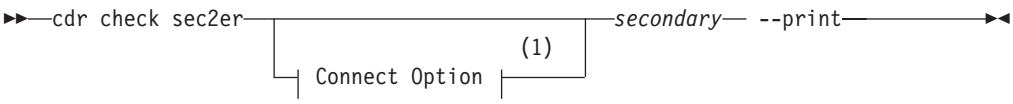
"cdr stats check" on page A-143

"cdr disable server" on page A-86

cdr check sec2er

The **cdr check sec2er** command determines whether a high availability cluster can be converted to replication servers.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
secondary	Name of the secondary server in the cluster.		"Long Identifiers" on page A-3

The following table describes the **cdr check sec2er** option.

Long Form	Short Form	Meaning
--print	**-p**	Shows the commands that would be run by the **cdr start sec2er** command during a conversion.

Usage

You must run the **cdr check sec2er** command from a primary server in a cluster with a high-availability data replication secondary or a remote stand-alone secondary server. The output of the **cdr check sec2er** command can show warning messages and error messages:

- Warning messages indicate possible problems for replication after the conversion. You can solve these problems after converting the cluster to replication servers.
- Error messages indicate problems preventing the conversion to replication server. You must solve all error conditions before you run the **cdr start sec2er** command to convert a cluster to replication servers.

Use the **--print** option to display the commands that are run during a conversion.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error code is returned: 225.

For information about these error codes, see "Return Codes for the cdr Utility" on page A-8.

Examples

The following example checks if a cluster consisting of a primary server named **priserv** and a secondary server named **secserv** can be converted to replication servers:

```
cdr check sec2er -c priserv secserv
```

The following output of the **cdr check sec2er** command indicates that conversion would be successful, but that several issues should be addressed either before or after conversion:

```
WARNING:CDR_SERIAL value on priserv can cause collisions.
WARNING:Dbspace is becoming full.
WARNING:Using the same values for CDR_SERIAL can cause collisions.

Secondary conversion to ER is possible.
Errors:0000   Warnings:0003
```

The following output of the **cdr check sec2er** command indicates that conversion will not be successful until the CDR_QDATA_SBSPACE configuration parameter is set in the `onconfig` file on both the primary and the secondary servers:

```
WARNING:CDR_SERIAL value on priserv can cause collisions.
WARNING:Dbspace is becoming full.
WARNING:Using the same values for CDR_SERIAL can cause collisions.
ERROR:ER sbspace not correctly set up (CDR_QDATA_SBSPACE).

Secondary conversion to ER is not possible.
Errors:0001   Warnings:0003
```

The following output of the **cdr check sec2er** command indicates that conversion will not be successful until the `sqlhosts` files on both the primary and the secondary servers are correctly configured for Enterprise Replication:

```
WARNING:CDR_SERIAL value on serv1 can cause collisions.
ERROR:Server priserv and server secserv belong to the same group.
WARNING:Dbspace  is becoming full.
ERROR:Server priserv and server secserv belong to the same group.
WARNING:Using the same values for CDR_SERIAL can cause collisions.
FATAL:SQLHOSTS is not set up correctly for ER.
ERROR:SQLHOSTS is not set up correctly for ER.
ERROR:ER sbspace not correctly set up (CDR_QDATA_SBSPACE).

Secondary conversion to ER is not possible.
Errors:0004   Warnings:0003
```

The following example shows the output of the **--print** option, which describes the commands that will be run when the **cdr start sec2er** command is run on the **priserv** server. The servers are defined as replication servers. Any tables that do not have a primary key are altered to add ERKEY shadow columns. A replicate is created and started for each user table on the **priserv** server.

```
$cdr check sec2er --print serv2
Secondary conversion to ER is possible.

Errors:0000   Warnings:0000
--
--  Define ER for the first time
--
cdr define serv -c cdr1 -I cdr1

--
--  Creating Replication Key
--
dbaccess - - <<EOF
database stores_demo;
alter table 'mpruet'.classes add ERKEY;
EOF

--
--  Define the replicates
--
--
--  Defining Replicates for Database stores_demo
--
cdr define repl --connect=cdr1 sec2er_1_1282611664_call_type --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.call_type" \
         "select * from 'mpruet'.call_type"
cdr start repl --connect=cdr1 sec2er_1_1282611664_call_type

cdr define repl --connect=cdr1 sec2er_4_1282611664_cust_calls --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.cust_calls" \
         "select * from 'mpruet'.cust_calls"
cdr start repl --connect=cdr1 sec2er_4_1282611664_cust_calls

cdr define repl --connect=cdr1 sec2er_5_1282611664_customer --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.customer" \
         "select * from 'mpruet'.customer"
cdr start repl --connect=cdr1 sec2er_5_1282611664_customer

cdr define repl --connect=cdr1 sec2er_3_1282611664_classes --master=cdr1 \
   --conflict=always --scope=row \
   "stores_demo@cdr1:'mpruet'.classes" \
         "select * from 'mpruet'.classes"
```

```
cdr start repl --connect=cdr1 sec2er_3_1282611664_classes
--
--  Starting RSS to ER conversion
--
--
--  WARNING:
--
--  DDL statements will not be automatically propagated to the ER server
--  after converting the secondary server into an ER server.  If you
--  create or alter any objects, such as databases, tables, indexes, and
--  so on, you must manually propagate those changes to the ER node and
--  change any replication rules affecting those objects.
--
```

Related reference:

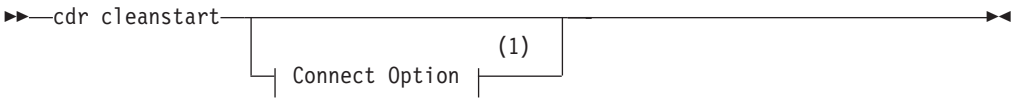
"cdr start sec2er" on page A-137

"Example of creating a new replication domain by cloning" on page 6-3

cdr cleanstart

The **cdr cleanstart** command starts an Enterprise Replication server with empty queues.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Usage

The **cdr cleanstart** command starts an Enterprise Replication server, but first empties replication queues of pending transactions. Use this command if synchronizing the server using the **cdr sync** command would be quicker than letting the queues process normally.

If an Enterprise Replication server was restored from a backup, but the restore did not include all log files from the replay position, or the system was not restored to the current log file, advance the log file unique ID past the latest log file unique ID prior to the restore, and then run the **cdr cleanstart** command followed by the **cdr sync** command to synchronize the server.

You can run this command from within an SQL statement by using the SQL administration API.

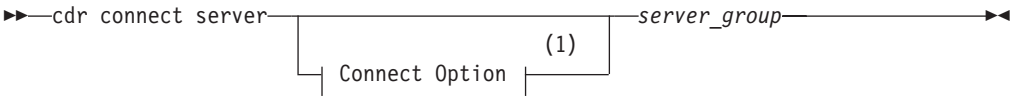
Related reference:

"cdr start" on page A-129

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



Notes:

1 See "Connect Option" on page A-3.

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.	"Long Identifiers" on page A-3

Usage

When you run the **cdr connect server** command, an event alarm with a class ID of 53 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Related reference:

"cdr define server" on page A-71

"cdr delete server" on page A-81

"cdr disconnect server" on page A-88

"cdr list server" on page A-103

"cdr modify server" on page A-112

"cdr resume server" on page A-128

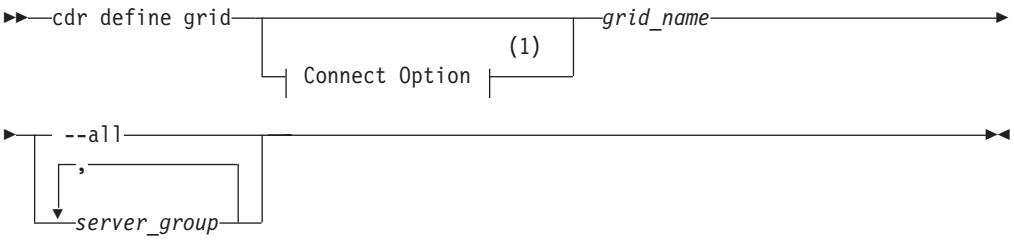
"cdr suspend server" on page A-158

"Enterprise Replication Event Alarms" on page 9-21

cdr define grid

The **cdr define grid** command creates a named grid of replication servers to simply administration.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be unique among grid names and replicate set names.	"Long Identifiers" on page A-3
server_group	Name of a database server group to add to the grid.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3

The following table describes the **cdr define grid** option.

Long Form	Short Form	Meaning
--all	**-a**	Include all replication servers in the domain.

Usage

You must run the **cdr define grid** command from a replication server that is a member of an Enterprise Replication domain.

Use the **--all** to include all replication servers in the domain in the grid.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 221.

For information on these error codes, see "Return Codes for the cdr Utility" on page A-8.

Examples

The following example defines a grid named **grid1** and adds two replication servers to it:

```
cdr define grid grid1 gserv1, gserv2
```

The following example defines a grid named **grid1** and adds all replication servers in the current domain to it:

```
cdr define grid grid1 --all
```

Related tasks:

"Creating a grid" on page 7-4

Related reference:

"cdr change grid" on page A-30

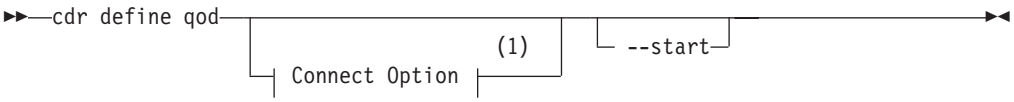
"cdr list grid" on page A-94

"cdr delete grid" on page A-78

cdr define qod

The **cdr define qod** command enables monitoring the quality of replicated data for the replication servers in a grid.

Syntax



Notes:

1 See "Connect Option" on page A-3.

The following table describes the **cdr define qod** option.

Long Form	Short Form	Meaning
--start	**-s**	Specifies to start monitoring data quality.

Usage

Use the **cdr define qod** command to define a master server for monitoring the quality of replicated data for the replications servers in a grid so that a Connection Manager can use this information to decide how to route client connections to replication servers based on the SLA. You must run the **cdr define qod** command from a root server.

The following policies of the Connection Manager SLA require that the monitoring of the quality of data be defined and started:

- FAILURE: Connection Manager selects the server with the fewest apply failures.
- LATENCY: Connection Manager selects the server with the lowest latency. Latency is a measure of how long it takes to replicate transactions.

You can start monitoring by including the **--start** option or by running the **cdr start qod** command.

If the monitoring for data quality is already enabled, running the **cdr define qod** command changes the master server.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 217.

For information on error codes, see "Return Codes for the cdr Utility" on page A-8.

Example

The following command defines the server being connected to, **gserv_2**, as the master server for data quality and starts the monitoring of data quality:

```
cdr define qod -C gserv_2 --start
```

Related tasks:

"Routing client connections in a grid" on page 7-16

Related reference:

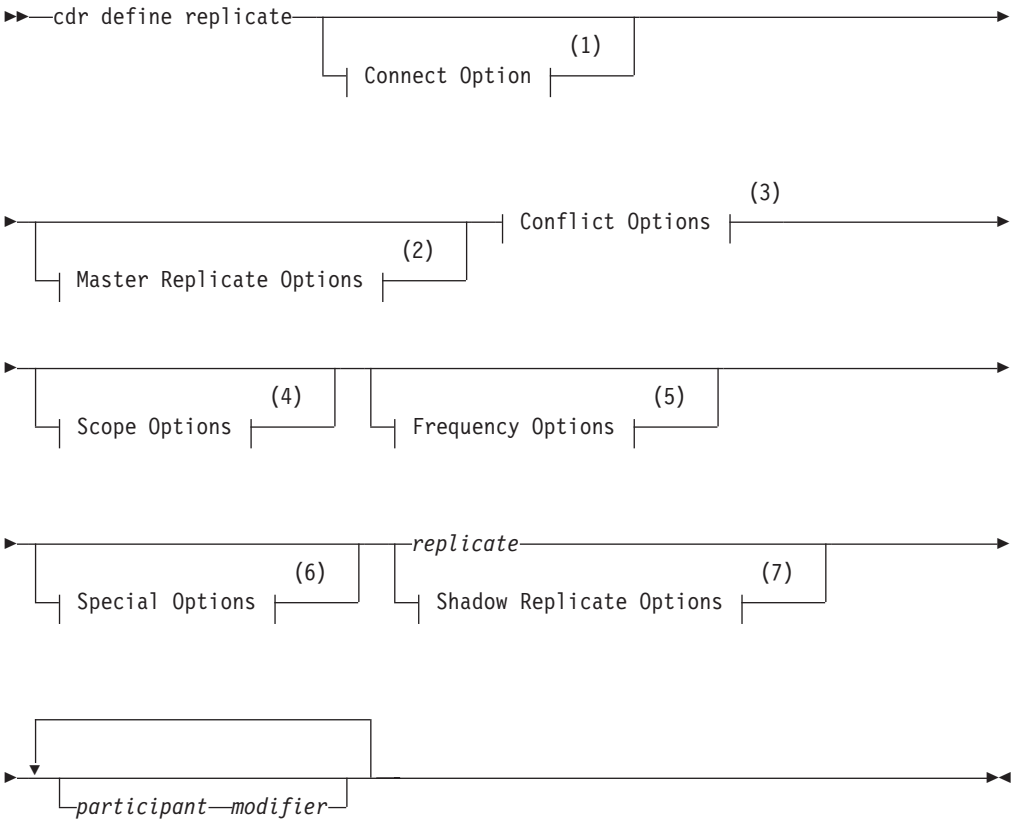
"cdr start qod" on page A-130

"cdr stop qod" on page A-152

cdr define replicate

The **cdr define replicate** command defines a replicate in the global catalog.

Syntax



Notes:

1 See "Connect Option" on page A-3.

2 See Master Replicate Options.

3 See "Conflict Options" on page A-63.

4 See "Scope Options" on page A-65.

5 See "Frequency Options" on page A-26.

6 See "Special Options" on page A-65.

7 See "Shadow Replicate Options" on page A-67.

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		"Participant and participant modifier" on page A-4
participant	Name of a participant in the replication.	The participant must exist.	"Participant and participant modifier" on page A-4
replicate	Name of the new replicate.	The replicate name must be unique.	"Long Identifiers" on page A-3

Usage

To be useful, a replicate must include at least two participants. You can define a replicate that has one or no participant, but before you can use that replicate, you must use the **cdr change replicate** command to add more participants. You cannot start and stop replicates that have no participants.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

When you define a replicate, the replicate does not begin until you explicitly change its state to *active* by running the **cdr start replicate** command.

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.

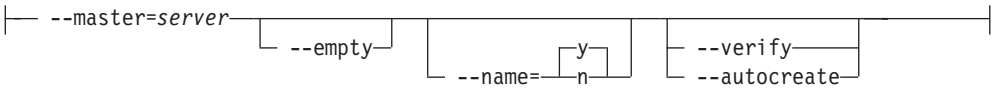
You can run this command from within an SQL statement by using the SQL administration API.

Master Replicate Options

The master replicate options specify whether Enterprise Replication defines the replicate as a master replicate. A master replicate uses saved dictionary information about the attributes of replicated columns to verify that participants conform to the specified schema. You must specify at least one participant when creating a master replicate. All participants specified are verified when the **cdr define replicate** or **cdr change replicate** command is run. If any participant does not conform to the

master definition, the command fails and that local participant is disabled. If a participant you specify does not contain the master replicate table, Enterprise Replication automatically creates the table on the participant based on the master replicate dictionary information. All database servers containing master replicates must be able to establish a direct connection with the master replicate database server.

Master Replicate Options:



Element	Purpose	Restrictions	Syntax
server	Name of the database server group from which to base the master replicate definition.	The name must be the name of a database server group.	"Long Identifiers" on page A-3

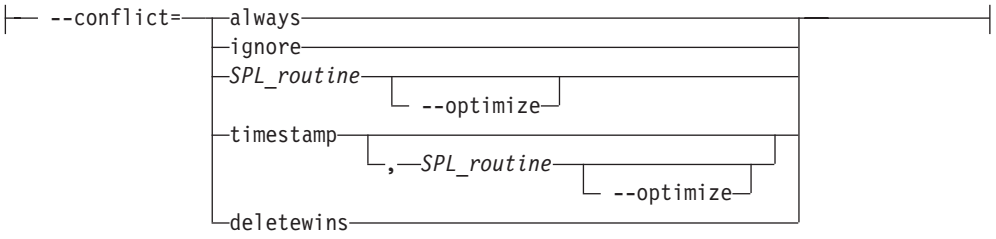
The following table describes the master replicate options.

Long Form	Short Form	Meaning
--master=	**-M**	Specifies that the replicate being created is a master replicate.
--empty	**-t**	Specifies that the participant on the server specified with the **--master** option is used as the basis of the master replicate, but is not added to the replicate.
--name=	**-n**	Specifies whether the master replicate has column name verification in addition to column data type verification. Valid values are: • **--name=y** = Default. Column names are verified to be the same on all participants. • **--name=n** = Column names are not verified and discrepancies can exist.
--verify	**-v**	Specifies that the **cdr define replicate** command verifies the database, tables, and column data types against the master replicate definition on all listed servers.
--autocreate	**-u**	Specifies that if the tables in the master replicate definition do not exist in the databases on the target servers, they are created automatically. However, the tables cannot contain columns with user-defined data types. The tables are created in the same dbspace as the database. **Note:** Tables created with the **--autocreate** option do not automatically include non-primary key indexes, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with the **--autocreate** option require the use of these objects you must manually create those objects.

Conflict Options

The **--conflict** options specify how Enterprise Replication resolves conflicts with data arriving at the database server.

Conflict Options:



Element	Purpose	Restrictions	Syntax
SPL_routine	SPL routine for conflict resolution	The SPL routine must exist.	"Long Identifiers" on page A-3

The following table describes the **--conflict** options.

Long Form	Short Form	Meaning
--conflict=	**-C**	Specifies the rule that is used for conflict resolution. • Use the **always** option if you do not want Enterprise Replication to resolve conflicts, but you do want replicated changes to be applied even if the operations are not the same on the source and target servers. Use the always-apply conflict resolution rule only with a primary-target replication system. If you use always-apply with an update-anywhere replication system, your data might become inconsistent. • Use the **ignore** option if you do not want Enterprise Replication to resolve conflicts. • Use the **timestamp** option to have the row or transaction with the most recent time stamp take precedence in a conflict. • Use the **deletewins** option to have the row or transaction with a DELETE operation, or otherwise with the most recent time stamp, take precedence in a conflict. The delete wins conflict resolution rule prevents upserts. The action that Enterprise Replication takes depends on 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 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. When this option is not present, Enterprise Replication always calls the SPL routine defined for the replicate when a conflict is detected.

Scope Options

The **--scope** options specify the scope of Enterprise Replication conflict resolution.

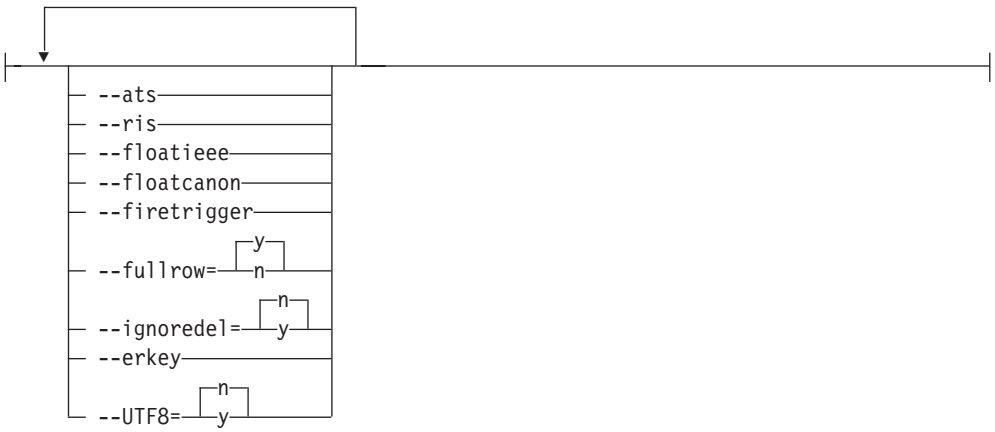
Scope Options:

The following table describes the **--scope** option.

Long Form	Short Form	Meaning
--scope=	**-S**	Specifies the scope that is invoked when Enterprise Replication encounters a problem with data or a conflict occurs. • **--scope=row** = Evaluate one row at a time and apply the replicated rows that win the conflict resolution with the target rows. • **--scope=transaction** = Default. Apply the entire transaction if the replicated transaction wins the conflict resolution. When specifying the scope, you can abbreviate **transaction** to **tra**.

Special Options

Special Options:



The following table describes the special options to **cdr define replicate**.

Long Form	Short Form	Meaning
--ats	**-A**	Activates aborted transaction spooling for replicate transactions that fail to be applied to the target database.

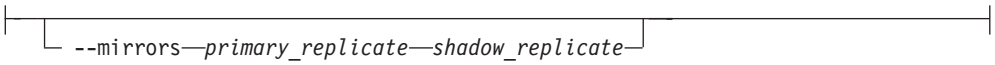
Long Form	Short Form	Meaning
--erkey	**-K**	Adds the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, to the participant definition, if the table contains the ERKEY shadow columns. The ERKEY shadow columns are used in place of a primary key. The **--erkey** option is not valid with the **cdr define template** command. The ERKEY shadow columns are automatically added to the replicate definition when you define a template.
--firetrigger	**-T**	Specifies that the rows that the replicate inserts fire triggers at the destination.
--floatieee	**-I**	Transfers replicated floating-point numbers in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format. Use this option for all new replicate definitions.
--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 compatibility with earlier versions only; use **--floatieee** for all new replicate definitions.
--fullrow=	**-f**	Specifies whether to replicate full rows or only the changed columns: • **--fullrow=y** = Default. Indicates to replicate the full row and to enable upserts. If you also specify **deletewins** as the conflict resolution rule, upserts are disabled. • **--fullrow=n** = Indicates to replicate only changed columns and disable upserts.
--ignoredel=	**-D**	Specifies whether to retain deleted rows on other nodes: • **--ignoredel=y** = Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify **deletewins** as the conflict resolution rule; the **cdr define replicate** command fails when these contradictory options are combined. • **--ignoredel=n** = Default. Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--ris	**-R**	Activates row-information spooling for replicate row data that fails conflict resolution or encounters replication order problems.

Long Form	Short Form	Meaning
--UTF8=	None	Specifies whether to enable conversion to and from UTF-8 (Unicode) when replicating data between servers that use different code sets. • **–UTF8=y** Default. Indicates that character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data contained within opaque data types such as UDT or DataBlade data. • **–UTF8=n** Indicates that code set conversion is ignored.

Shadow Replicate Options

A shadow replicate is a copy of an existing, or primary, replicate. You must create a shadow replicate to perform a manual remastering of a replicate that was defined with the **-n** option. After creating the shadow replicate, the next step in manual remastering is to switch the primary replicate and the shadow replicate using the **cdr swap shadow** command.

Shadow Replicate Options:



Element	Purpose	Restrictions	Syntax
primary_replicate	Name of the replicate on which to base the shadow replicate.	The replicate must exist. The replicate name must be unique.	"Long Identifiers" on page A-3
shadow_replicate	Name of the shadow replicate to create.	The replicate name must be unique.	"Long Identifiers" on page A-3

The following table describes the shadow replicate option to **cdr define replicate**.

Long Form	Short Form	Meaning
--mirrors	**-m**	Specifies that the replicate created is a shadow replicate based on an existing primary replicate.

Examples

Example 1: Define a replicate with two participants

The following example defines a replicate with two participants:

```
cdr define repl --conflict=timestamp,sp1 \
--scope=tran --ats --fullrow=n --floatieee newrepl \
"db1@iowa:antonio.table1" "select * from table1" \
"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** is run 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 Replication replicates only 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"**.

Example 2: Define a replicate with a frequency of every five hours

This 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 specifies that the data is replicated every five hours.

```
cdr def repl -c ohio -C timestamp,sp1 \
-S tran -A -e 5:00 -I newrepl \
"db1@iowa:antonio.table1" "select * from table1" \
"db2@utah:carlo.table2" "select * from table2"
```

Example 3: Define a master replicate

The following example defines a master replicate:

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl \
"db1@iowa:antonio.table1" "select * from table1"
```

Line 1 instructs Enterprise Replication to create a master replicate based on the replicate information from the database server **iowa**. Line 2 specifies the delete wins conflict resolution rule, a transaction scope, and that the name of the replicate is **newrepl**. Line 3 specifies the table and columns included in the master replicate.

Example 4: Define a master replicate and create a table on a participant

This example is the same as the previous example except that it specifies a second participant in Line 4. The second participant (**utah**) does not have the table **table1** specified in its participant and modifier syntax. The **-u** option specifies to create the table **table1** on the **utah** server.

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl -u\
"db1@iowa:antonio.table1" "select * from table1 \
"db2@utah:carlo.table1" "select * from table1"
```

Example 5: Define a replicate with the ERKEY shadow columns

This example defines a master replicate similar to the one in example 3, but also includes the ERKEY shadow columns that are used in place of a primary key.

```
cdr def repl -c iowa -M iowa \
-C deletewins -S tran newrepl --erkey\
"db1@iowa:antonio.table1" "select * from table1"
```

Related concepts:

"Conflict Resolution" on page 3-6
"Conflict Resolution Scope" on page 3-14
"Failed Transaction (ATS and RIS) Files" on page 9-3
"Frequency Options" on page A-26
"Preparing for Role Separation (UNIX)" on page 4-22

Related tasks:

"Enabling ATS and RIS File Generation" on page 9-4
"Specifying Conflict Resolution Rules and Scope" on page 6-10
"Replicating Only Changed Columns" on page 6-12
"Defining Master Replicates" on page 6-8
"Enabling Triggers" on page 6-13
"Using the IEEE Floating Point or Canonical Format" on page 6-13
"Defining Replication Servers" on page 6-1
"Setting Up Failed Transaction Logging" on page 6-11
"Preparing tables without primary keys" on page 4-21

Related reference:

"cdr change replicate" on page A-32
"cdr change replicateset" on page A-35
"cdr delete replicate" on page A-79
"cdr list replicate" on page A-97
"cdr modify replicate" on page A-108
"cdr resume replicate" on page A-126
"cdr start replicate" on page A-131
"cdr stop replicate" on page A-152
"cdr suspend replicate" on page A-155
"cdr swap shadow" on page A-159
"cdr define template" on page A-75
"cdr delete replicateset" on page A-80
"cdr list replicateset" on page A-101
"cdr modify replicateset" on page A-111
"cdr resume replicateset" on page A-127
"cdr start replicateset" on page A-134
"cdr stop replicateset" on page A-154
"cdr suspend replicateset" on page A-157
"Participant and participant modifier" on page A-4

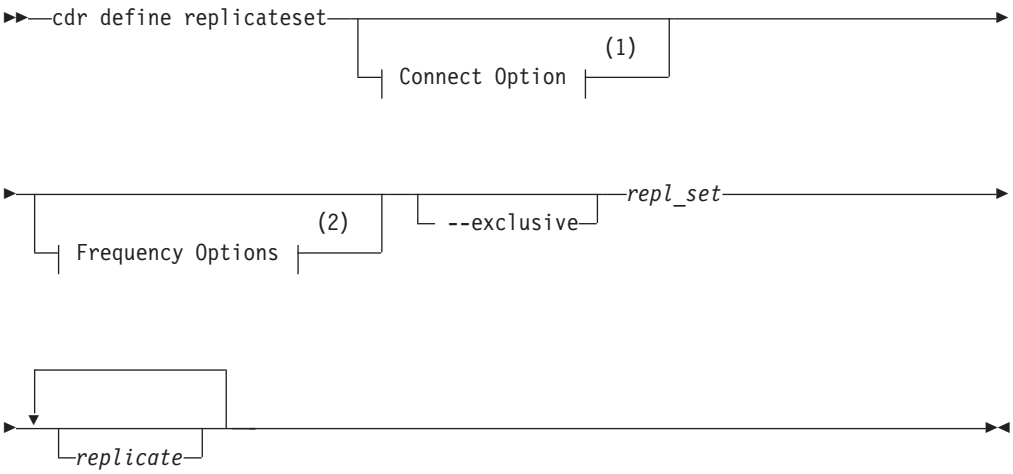
cdr define replicateset

The **cdr define replicateset** 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 IBM Informix, 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



Notes:

1 See "Connect Option" on page A-3.

2 See "Frequency Options" on page A-26.

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-3
replicate	Name of a replicate to be included in the replicate set.	The replicate must exist.	"Long Identifiers" on page A-3

The following table describes the option to **cdr define replicateset**.

Long Form	Short Form	Meaning
--exclusive	**-X**	Creates an exclusive replicate set.

Usage

Use the **cdr define replicateset** command to define a replicate set.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

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.

When you create an exclusive replicate set, the state is initially set to active.

To create an exclusive replicate set and make it active

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. You cannot change whether a replicate set is exclusive or not.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server and defines the non-exclusive replicate set **accounts_set** with replicates **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 replicates **basil** and **thyme**:

```
cdr def replset --connect=olive --exclusive market_set basil thyme
```

Related concepts:

"Frequency Options" on page A-26

"Preparing for Role Separation (UNIX)" on page 4-22

Related tasks:

"Exclusive Replicate Sets" on page 6-18

"Defining Replicate Sets" on page 6-18

Related reference:

"cdr change replicateset" on page A-35

"cdr delete replicateset" on page A-80

"cdr list replicateset" on page A-101

"cdr modify replicateset" on page A-111

"cdr resume replicateset" on page A-127

"cdr start replicateset" on page A-134

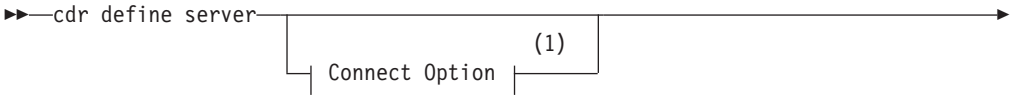
"cdr stop replicateset" on page A-154

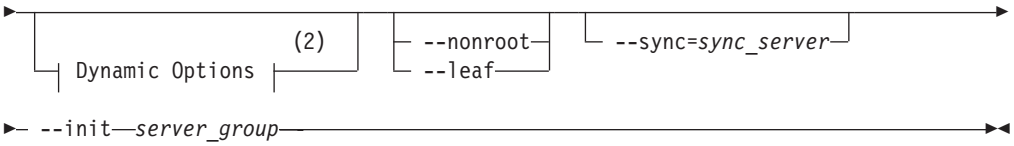
"cdr suspend replicateset" on page A-157

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





Notes:

1 See "Connect Option" on page A-3.

2 See "Dynamic Options."

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to add to an Enterprise Replication domain.	Must be the name of an existing database server group in SQLHOSTS.	
sync_server	Name of server to use to synchronize the global catalog of the new server with the other servers in the domain. Becomes the parent server of nonroot and leaf servers.	Must be a server that is registered with Enterprise Replication. The server must be online.	"Long Identifiers" on page A-3

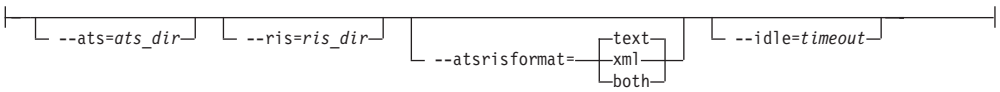
The following table describes the options to **cdr define server**.

Long Form	Short Form	Meaning
--init	**-I**	Adds *server_group* to the replication domain. The *server_group* must be the same as the connection server.
--leaf	**-L**	Defines the server as a leaf server in an existing domain. The server specified by the **--sync** option becomes the parent of the leaf server.
--nonroot	**-N**	Defines the server as a nonroot server in an existing domain. The server specified by the **--sync** option becomes the parent of the nonroot server.
--sync=	**-S**	Adds a server to an existing domain. Uses the global catalog on *sync_server* as the template for the global catalog on the new replication server, *server_group*. For Hierarchical Routing topologies, Enterprise Replication also uses the *sync_server* as the parent of the new server in the current topology.

Dynamic Options

The options allow you to modify the default behavior of **cdr define server** that you can change with **cdr modify server**.

Options:



Element	Purpose	Restrictions	Syntax
ats_dir	Name of the directory for Aborted Transaction Spooling files. The default is **/tmp**.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of **/dev/null** (UNIX) or **NUL** (Windows) prevents ATS file generation.	Follows naming conventions on your operating system
ris_dir	Name of the directory for Row Information Spooling files. The default is **/tmp**.	Must be a full path name. The path for the directory can be no longer than 256 characters. A value of **/dev/null** (UNIX) or **NUL** (Windows) prevents RIS file generation.	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 32767.	Integer

The following table describes the options to **cdr define server** that you can change with **cdr modify server**.

Long Form	Short Form	Meaning
--ats=	**-A**	Specifies the directory to store aborted transaction spooling files for replicate transactions that fail to be applied.
--atsrisformat=	**-X**	Specifies the format of ATS and RIS files: • **text** indicates that ATS and RIS files are generated in standard text format. • **xml** indicates that ATS and RIS files are generated in XML format. • **both** indicates that ATS and RIS files are generated in both standard text format and XML format. If you omit the **--atsrisformat=** option, ATS and RIS files are created in text format.
--ris=	**-R**	Specifies the directory to store row information spooling files for replicate row data that fails conflict resolution or encounters replication-order problems.
--idle=	**-i**	Set the number of minutes after which an inactive connection is terminated after *timeout* minutes. If timeout is `0`, the connection does not timeout. The default value is `0`.

Usage

The **cdr define server** command creates the Enterprise Replication global catalog and adds the database server from the *server_group*. If you specify a synchronization server, the new server is added to an existing Enterprise Replication domain. If you do not specify a synchronization server, Enterprise Replication creates a domain.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The first example defines the first database server in a replication domain. The command connects to the database server **stan**, initializes Enterprise Replication, and sets the idle timeout to 500 minutes. The example also specifies that ATS files are generated into the **/cdr/ats** directory, RIS files are generated in the **/cdr/ris** directory, and that the format of ATS and RIS files is text.

```
cdr define server --connect=stan \
--idle=500 --ats=/cdr/ats --ris=/cdr/ris \
--atsrisformat=text --init g_stan
```

The following example adds a database server to the replication domain that was created 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 sets an idle timeout of 600 minutes for the database server **oliver**. This command also specifies that ATS files are generated into the **/cdr/ats** directory, no RIS files is generated, and the ATS file format is XML.

```
cdr define server -c oliver -i 600 \
-A /cdr/ats -R /dev/null -X xml \
-S g_stan -I g_oliver
```

Related concepts:

Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1

"Database Server Groups" on page 4-2

Related tasks:

"Enabling ATS and RIS File Generation" on page 9-4

"Disabling ATS and RIS File Generation" on page 9-13

"Defining Replication Servers" on page 6-1

"Customizing the Replication Server Definition" on page 6-6

Related reference:

"cdr connect server" on page A-57

"cdr delete server" on page A-81

"cdr disconnect server" on page A-88

"cdr list server" on page A-103

"cdr modify server" on page A-112

"cdr resume server" on page A-128

"cdr suspend server" on page A-158

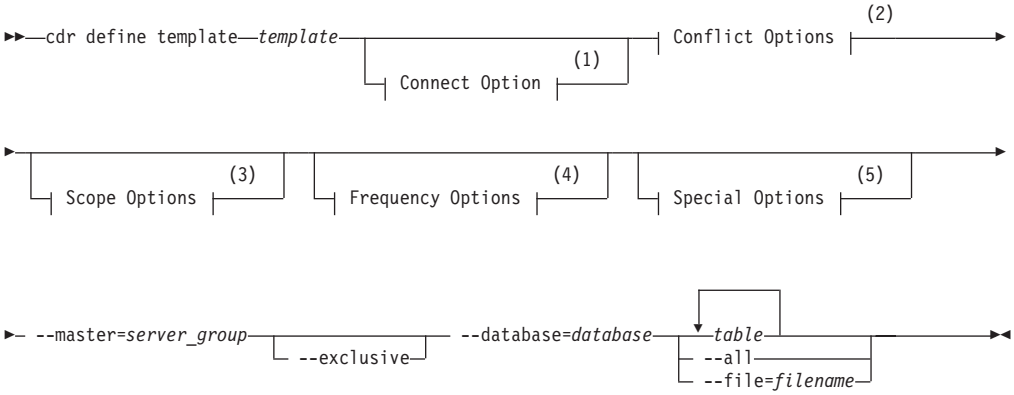
"cdr realize template" on page A-114

cdr define template

The **cdr define template** command creates a template for replicates and a replicate set.

Because templates define replicates, many of the syntax options for the **cdr define template** command are the same as for the **cdr define replicate** command.

Syntax



Notes:

1 See "Connect Option" on page A-3.

2 See "Conflict Options" on page A-63.

3 See "Scope Options" on page A-65.

4 See "Frequency Options" on page A-26.

5 See "Special Options" on page A-65.

Element	Purpose	Restrictions	Syntax
template	Name of the template to create.	The template name must be unique and cannot be the same as a replicate or replicate set name.	"Long Identifiers" on page A-3
database	Name of the database used to define the template attributes.	The database server must be registered with Enterprise Replication.	"Long Identifiers" on page A-3
table	Name of the table to be included in the template.	The table must be an actual table. It cannot be a synonym or a view. For ANSI databases, you must specify *owner.tablename*.	"Long Identifiers" on page A-3
filename	The directory and filename of the file containing a list of tables to be included in the template.	Must be a full pathname and filename. The path and filename can be no longer than 256 bytes. Within the file, the table names can be separated by a space or placed on different lines.	Follows naming conventions on your operating system.
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.	"Long Identifiers" on page A-3

The following table describes the options to **cdr define template**.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all tables in the database are included in the template.
--database=	**-d**	Specifies which database the template is based on. If no tables or table list filename are listed after this option, then all tables in the database are included in the template.
--exclusive	**-X**	Creates an exclusive replicate set. The state of the replicate set is inactive until you apply the template. This option is required if you have referential integrity constraints on a table.
--file=	**-f**	Specifies the path and filename of a file that lists the tables to be included in the template. The file must contain only table names, either separated by spaces or each on its own line.
--master=	**-M**	Specifies the server that contains the database to be used as the basis of the template. If this option is not specified, then the server specified in the connect option is used.

Usage

A template consists of schema information about a database, a group of tables, column attributes, and the primary keys that identify rows. A template defines a group of master replicates and a replicate set. Templates are an alternative to using

the **cdr define replicate** and **cdr start replicate** commands for each table and manually combining the replicates into a replicate set by using the **cdr define replicateset** command.

The replicate set can be exclusive or non-exclusive. Specify that the replicate set is exclusive if you have referential constraints placed on the replicated columns. If you create an exclusive replicate set using a template, you do not need to stop the replicate set to add replicates. The **cdr define template** command performs this task automatically.

If your tables include the ERKEY shadow columns, they are automatically added to replicate definition when you define a template. The **--erkey** option is not needed with the **cdr define template** command.

You cannot specify an SPL routine for conflict resolution when you define a template.

After you define a template using the **cdr define template** command, use the **cdr realize template** command to apply the template to your Enterprise Replication database servers.

You cannot update a template. To modify a template, you must delete it and then re-create it with the **cdr define template** command.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example illustrates the **cdr define template** command:

```
cdr define template tem1 -c detroit\
-C timestamp -S tran \
--master=chicago\
--database=new_cars table1 table2 table3
```

Line 1 of the example specifies a template name of **tem1** and that the connection is made to the server **detroit**. Line 2 specifies a conflict-resolution rule of **timestamp** and a transaction scope for conflict resolution. Line 3 specifies that the master replicate information is obtained from the server **chicago**. Line 4 specifies to use the **new_cars** database on the **chicago** server and to include only the tables **table1**, **table2**, and **table3**.

The next example is the same as the first except that it has additional options and uses a file instead of a list of tables:

```
cdr define template tem1 -c detroit\
-C timestamp -S tran --master=chicago\
--ignoredel=y\
--database=new_cars --file=tabfile.txt
```

Line 3 indicates that delete operations are not replicated. Retaining deleted rows on target servers is useful for consolidation models.

Line 4 specifies a filename for a file that contains a list of tables to include in the template. The **tabfile.txt** file has the following contents:

```
table1
table2
table3
table4
```

Related concepts:

"Database Server Groups" on page 4-2
"Frequency Options" on page A-26
"Preparing for Role Separation (UNIX)" on page 4-22

Related tasks:

"Exclusive Replicate Sets" on page 6-18
"Defining Master Replicates" on page 6-8
"Preparing tables without primary keys" on page 4-21
"Creating replicated tables through a grid" on page 7-10

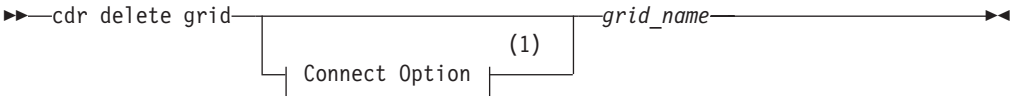
Related reference:

"cdr list template" on page A-106
"cdr realize template" on page A-114
"cdr delete template" on page A-84
"cdr define replicate" on page A-61
"Participant and participant modifier" on page A-4

cdr delete grid

The **cdr delete grid** command deletes the specified grid.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	"Long Identifiers" on page A-3

Usage

Use the **cdr enable grid** command to delete an existing grid.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 222.

For information about these error codes, see "Return Codes for the cdr Utility" on page A-8.

Examples

The following example deletes the grid named **grid1**:

```
cdr delete grid grid1
```

Related reference:

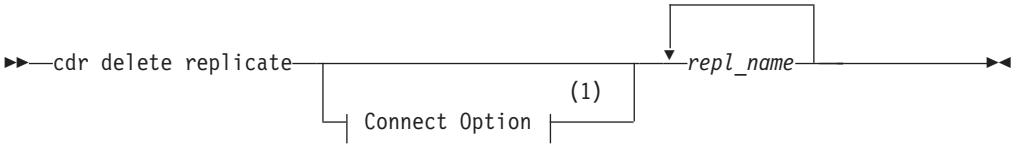
"ifx_grid_connect() procedure" on page C-1

"cdr define grid" on page A-58

cdr delete replicate

The **cdr delete replicate** command deletes a replicate.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate to delete.	The replicate must exist.	"Long Identifiers" on page A-3

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.

Important: Avoid deleting a replicate and immediately recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later.

When you run the **cdr delete replicate** command, an event alarm with a class ID of 68 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following command connects to the default database server specified by the **INFORMIXSERVER** environment variable and deletes the replicate **smile**:

```
cdr del rep smile
```

Related tasks:

"Deleting a Replicate" on page 8-9

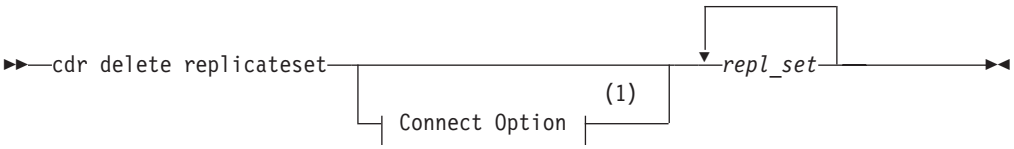
Related reference:

"cdr change replicate" on page A-32
"cdr define replicate" on page A-61
"cdr list replicate" on page A-97
"cdr modify replicate" on page A-108
"cdr resume replicate" on page A-126
"cdr start replicate" on page A-131
"cdr stop replicate" on page A-152
"cdr suspend replicate" on page A-155
"Enterprise Replication Event Alarms" on page 9-21

cdr delete replicateset

The **cdr delete replicateset** command deletes a replicate set.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to delete.	The replicate set must exist.	"Long Identifiers" on page A-3

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.

Attention: Do not delete time-based exclusive replicate sets. Doing so might result in inconsistent data.

Attention: Avoid deleting a replicate set and immediately recreating it with the same name. If you recreate the objects immediately (before the operation finishes propagating to the other Enterprise Replication database servers in the network), failures might occur in the Enterprise Replication system at the time of the operation or later.

When you run the **cdr delete replicateset** command, an event alarm with a class ID of 69 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server and deletes the replicate set **accounts_set**:

```
cdr del replset accounts_set
```

Related tasks:

"Deleting a Replicate Set" on page 8-13

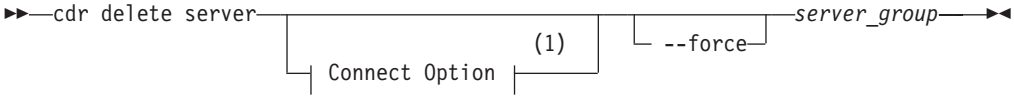
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr define replicate" on page A-61
"cdr list replicateset" on page A-101
"cdr modify replicateset" on page A-111
"cdr resume replicateset" on page A-127
"cdr start replicateset" on page A-134
"cdr stop replicateset" on page A-154
"cdr suspend replicateset" on page A-157
"Enterprise Replication Event Alarms" on page 9-21

cdr delete server

The **cdr delete server** command deletes a database server.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server group to remove from the global catalog.	The database server group must be currently defined in Enterprise Replication.	"Long Identifiers" on page A-3

The following table describes the option to the **cdr delete server** command.

Long Form	Short Form	Meaning
--force	**-f**	Specifies to remove the replication server from the global catalog while Enterprise Replication is not active. You must use this option if you are removing Enterprise Replication from a high-availability cluster server that was converted to a standard server.

Usage

Use the **cdr delete server** command to disable the ability of the server to participate in Enterprise Replication. If the replication server is inactive, use the **--force** option to remove Enterprise Replication capability. You cannot delete a server that has non-root or leaf children under it. You must delete the children of a server before deleting the parent server.

To remove an Enterprise Replication server from a domain, you must run the **cdr delete server** command *twice*:

1. On the server being removed, to remove Enterprise Replication from that server. This command is not propagated to other servers in the domain.
2. On another server, to remove the deleted server from the domain.

To remove an entire Enterprise Replication domain, run the **cdr delete server** command once on each replication server.

On the server being deleted from Enterprise Replication, the **cdr delete server** command performs the following tasks:

1. Drops the Enterprise Replication connection to other hosts in the domain. A 25582 error and an operating system error might be printed to the online log.
2. Removes Enterprise Replication information, including delete tables and shadow columns.
3. Shuts down Enterprise Replication, if it is running.
4. Drops the local copy of the global catalog (**syscdr**).

When you run the **cdr delete server** command on another root server in the domain, the command performs the following tasks:

1. Deletes the database server from the global catalogs of all other servers in the domain.
2. Removes the database server from all participating replicates.
3. Purges all replication data destined for the deleted server from the send queues.

Use the **cdr delete server –force** command to remove Enterprise Replication from a high-availability cluster server after it was converted to a standard server using the **oninit -d standard** command.

Attention: Do not delete a replication server and immediately re-create 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 domain), failures might occur in the Enterprise Replication system at the time of the operation or later.

When you run the **cdr delete server** command, event alarms with class IDs of 67 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

Example 1: Removing a single database server from the domain

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 -c italy g_italy
cdr delete server -c usa g_italy
```

The first 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.

The second 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

Example 2: Removing the whole domain

The following illustration shows a replication environment with three replication servers, **g_usa**, **g_italy**, and **g_japan**.

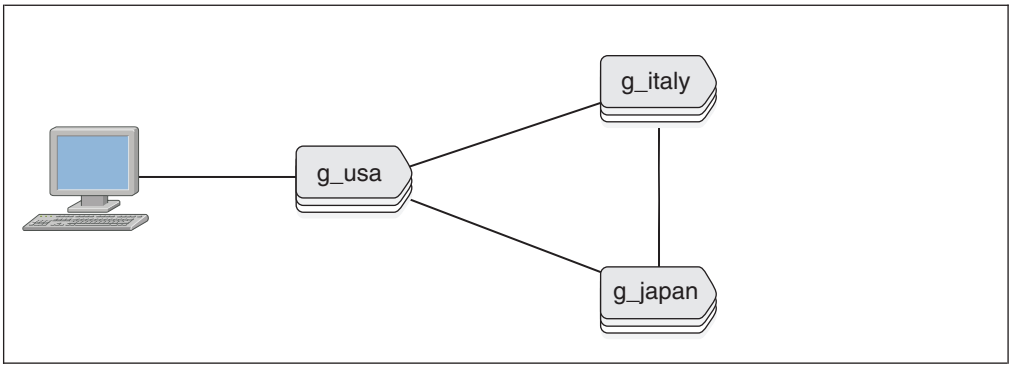


Figure A-1. Three Replication Servers

To remove Enterprise Replication from every server in the domain, issue the **cdr delete server** command while connecting to each server. For example, from the computer of the **g_usa** replication server, run these commands to remove Enterprise Replication and eliminate the domain:

```
cdr delete server -c italy g_italy
cdr delete server -c japan g_japan
cdr delete server g_usa
```

Example 3: Removing Enterprise Replication from a high-availability server

In this example, the replication server group **g_usa** contains two servers that participate in a high-availability cluster: a primary and a secondary (**usa_s**). After converting **usa_s** to a stand-alone server, the following command removes Enterprise Replication from it:

```
cdr delete server -c usa_s -f g_usa
```

Related tasks:

"Failure of the Primary Server in a High-Availability Cluster" on page 5-6

"Deleting a Replication Server" on page 8-5

Related reference:

"cdr connect server" on page A-57

"cdr define server" on page A-71

"cdr disconnect server" on page A-88

"cdr list server" on page A-103

"cdr modify server" on page A-112

"cdr resume server" on page A-128

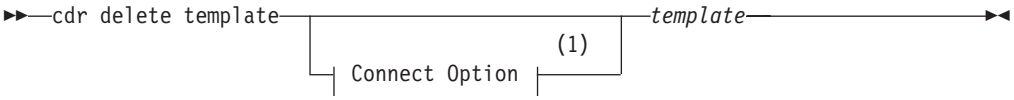
"cdr suspend server" on page A-158

"Enterprise Replication Event Alarms" on page 9-21

cdr delete template

The **cdr delete template** command deletes a template from the replication domain. It also deletes any underlying replicate sets associated with the template (these will exist if the template has been realized). No replicates are deleted.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
template	Name of the template to delete.	The template must exist.	"Long Identifiers" on page A-3

Usage

Use the **cdr delete template** command to delete the template definition and the replicate set realized from the template. Any replicates created by realizing the template to a database server are unaffected by this command.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following command deletes the template and replicate set **tem1**:

```
cdr delete template tem1
```

Related reference:

"cdr define template" on page A-75

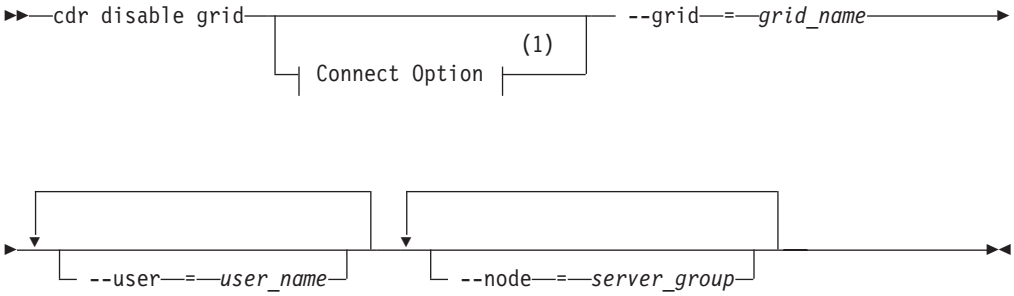
"cdr realize template" on page A-114

"Enterprise Replication Event Alarms" on page 9-21

cdr disable grid

The **cdr disable grid** command removes the authorization to run grid routines from users or servers.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	"Long Identifiers" on page A-3
server_group	Name of a database server group in the grid.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
user_name	Name of the user.	Must be a user with authorization to run grid routines.	"Long Identifiers" on page A-3

The following table describes the **cdr disable grid** options.

Long Form	Short Form	Meaning
--grid=	**-g**	Specifies the grid for which to revoke privileges.
--node=	**-n**	Specifies the servers on which to revoke privileges.
--user=	**-u**	Specifies the users to revoke privileges.

Usage

Use the **cdr disable grid** command to revoke the permission to run routines on the specified grid from the specified user or server that were granted by the **cdr enable grid** command.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see "Return Codes for the cdr Utility" on page A-8.

Examples

The following example revokes privileges from user **bill** on the **grid1** grid:

```
cdr disable grid --grid=grid1 --user=bill
```

The following example shows how to change the authorized server on the **grid1** grid from **gserv1** to **gserv2**:

```
cdr disable grid --grid=grid1 --node=gserv1
cdr enable grid --grid=grid1 --node=gserv2
```

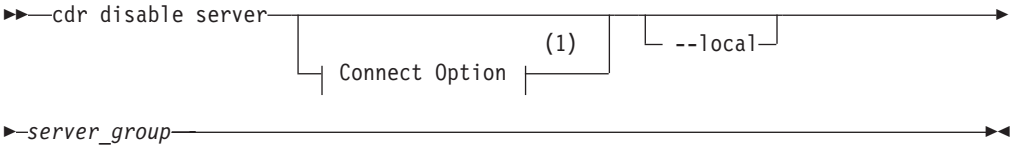
Related tasks:

"Maintaining the grid" on page 7-5

cdr disable server

The **cdr disable server** command disables replication on a server.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server on which to disable replication.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3

The following table describes the **cdr disable server** option.

Long Form	Short Form	Meaning
--local	**-l**	Disables the specified replication server. Must be run on both the replication server to disable and another replication server in the domain. Use this option if the connection is down between the replication server to disable and other replication servers.

Usage

Use the **cdr disable server** command when you need to temporarily stop replication and your replicates use the time stamp or delete wins conflict resolution rule.

When you run the **cdr disable server** command, the replication server is disabled and the rest of the replication domain is notified that the server is disabled.

If the replication server that you want to disable is not connected to the replication domain, you must run the **cdr disable server** command with the **--local** option on both the replication server to disable and another replication server in the domain. If the server on which you need to disable replication is currently offline, then run the **cdr disable server** command with the **--local** option on it after you restart it.

Disabling replication has the following effects:

- There is no connection between the disabled replication server and active replication servers.
- Transactions on the disabled replication server are not queued for replication
- Transactions on active replication servers are not queued for the disabled replication server.
- Control messages on active replication server are queued for the disabled replication server.
- Information about deleted rows on the disabled replication server is saved in delete tables.
- You can run only the following Enterprise Replication commands on the disabled replication server:
  - **cdr enable server**
  - **cdr stop server**
  - **cdr delete server**
  - **cdr check replicateset** with the **--repair** and the **--enable** options

You must synchronize the server after you enable replication on it. Shutting down and restarting the disabled replication server does not enable replication. You can both enable and synchronize a disabled replication server by running the **cdr check replicateset** command with the **--repair** and the **--enable** options. Alternatively, you can run the **cdr enable server** command and then synchronize the server.

Examples

The following examples show how to disable replication on a server.

Example 1: Stopping replication on a connected server

The following command disables the server, **g_cdr1**, which is connected to the replication domain:

```
cdr disable server -c g_cdr1 g_cdr1
```

Example 2: Stopping replication on a disconnected server

The following commands disable the replication server, **g_cdr1**, which is not connected to the replication domain:

```
cdr disable server -c g_cdr1 --local g_cdr1
cdr disable server -c g_cdr2 --local g_cdr1
```

The first command runs on the server **g_cdr1** and disables replication on it. The second command runs on the server **g_cdr2** and stops the other servers in the replication domain from queuing transactions for the server **g_cdr1**.

Related concepts:

"Time Stamp Conflict Resolution Rule" on page 3-7

"Delete Wins Conflict Resolution Rule" on page 3-12

Related tasks:

"Temporarily stopping replication on a server" on page 8-3

Related reference:

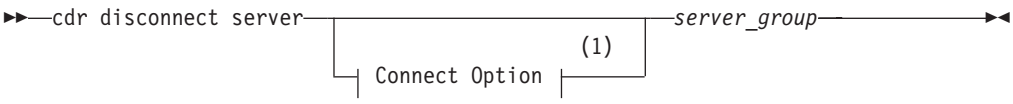
"cdr enable server" on page A-90

"cdr check replicateset" on page A-47

cdr disconnect server

The **cdr disconnect server** command stops a server connection.

Syntax



Notes:

1 See "Connect Option" on page A-3.

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.	"Long Identifiers" on page A-3

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 (the one specified by the **INFORMIXSERVER** environment variable).

When you run the **cdr disconnect server** command, event alarms with class IDs of 54 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example drops the connection between the default database server (the one specified by the **INFORMIXSERVER** environment variable) and the server group **g_store1**:

```
cdr disconnect server g_store1
```

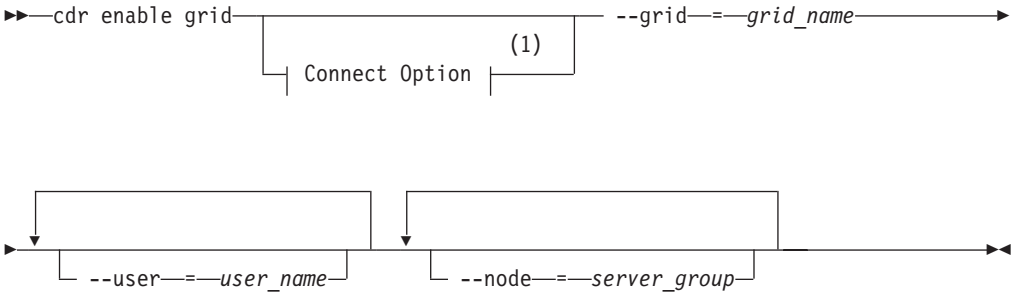
Related reference:

"cdr connect server" on page A-57
"cdr define server" on page A-71
"cdr delete server" on page A-81
"cdr list server" on page A-103
"cdr modify server" on page A-112
"cdr resume server" on page A-128
"cdr suspend server" on page A-158

cdr enable grid

The **cdr enable grid** command authorizes users to run commands on the grid and designates servers from which grid commands can be run.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	"Long Identifiers" on page A-3
server_group	Name of a database server group in the grid.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
user_name	Name of the user.	Must have Connect privilege for databases on all the replication servers in the grid.	"Long Identifiers" on page A-3

The following table describes the **cdr enable grid** options.

Long Form	Short Form	Meaning
--grid=	**-g**	Specifies the grid for which to provide privileges.
--node=	**-n**	Specifies the servers on which to provide privileges.
--user=	**-u**	Specifies the users to provide privileges.

Usage

Use the **cdr enable grid** command to control who can perform grid operations from which server in the grid. All the authorized users can run grid commands on all the authorized servers. The users must have Connect privilege for all databases on which they run grid routines on all the servers in the grid. You must authorize at least one user and one server to be able to run commands from the grid. User **informix** does not have permission to perform grid operations unless you include it in the user list.

Authorizing more than one server from which to run grid commands can lead to conflicts between grid commands.

After you initially enable a grid, you can add authorized users and servers by running the **cdr enable grid** command with the appropriate options.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see "Return Codes for the cdr Utility" on page A-8.

Examples

The following example authorizes the users **bill** and **tom** and the server **gserv1** to run grid routines on the grid named **grid1**:

```
cdr enable grid --grid=grid1 --user=bill --user=tom --node=gserv1
```

The following example adds the user **srini** to the list of authorized users for the **grid1** grid:

```
cdr enable grid --grid=grid1 --user=srini
```

The following example adds the server **gserv2** to the list of authorized servers for the **grid1** grid:

```
cdr enable grid --grid=grid1 --node=gserv2
```

Related concepts:

"Example of setting up a replication system with a grid" on page 7-2

Related tasks:

"Creating a grid" on page 7-4

"Maintaining the grid" on page 7-5

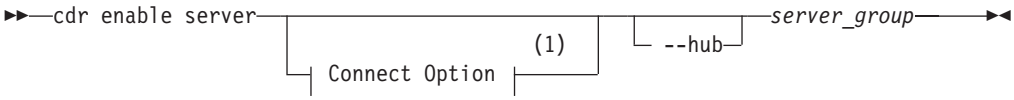
Related reference:

"ifx_grid_connect() procedure" on page C-1

cdr enable server

The **cdr enable server** command enables replication on a replication server that was disabled by the **cdr disable server** command.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
server_group	Name of the database server to enable.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3

The following table describes the **cdr enable server** option.

Long Form	Short Form	Meaning
--hub	**-h**	Specifies that when replication on a hub server is enabled, replication on all its child servers is also enabled.

Usage

Use the **cdr enable server** command when you are ready to restart replication on a disabled replication server. After you enable replication, you must synchronize the server with the rest of the replication domain. Before synchronization is complete, the replicates on the newly enabled replication server have the Pending Sync attribute. For replicates with the Pending Sync attribute, ATS and RIS files are not created if transactions are aborted on this server. You can see the Pending Sync attribute of a replicate in the OPTIONS field of the output of the **cdr list replicate** command.

Examples

The following command enables the disabled replication server, **g_cdr1**:

```
cdr enable server -c g_cdr1 g_cdr1
```

The following command enables the disabled replication server, **g_cdr1**, and its child servers:

```
cdr enable server -c g_cdr1 --hub g_cdr1
```

Related tasks:

"Restarting Replication on a Server" on page 8-4

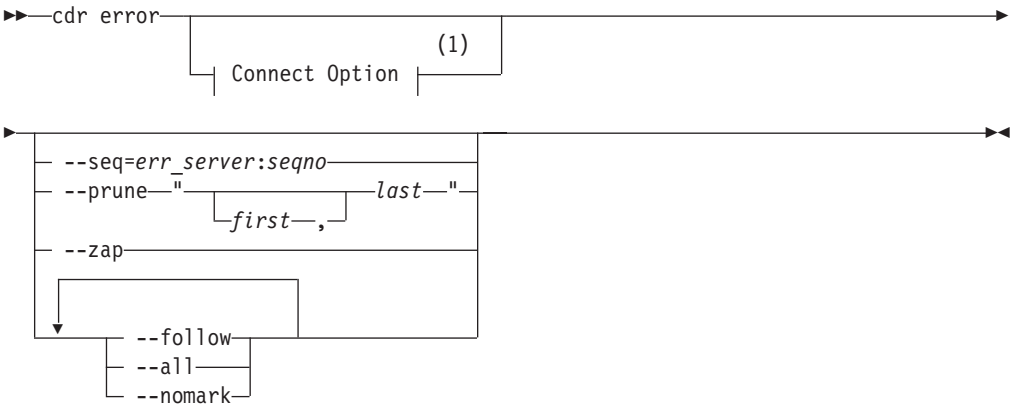
Related reference:

"cdr disable server" on page A-86

cdr error

The **cdr error** command manages the **syscdrerror** table and provides convenient displays of errors.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Table A-3. Elements for the cdr error command

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-3
first	Start date for a range.	You must provide a valid date and time.	"Frequency Options" on page A-26
last	Ending date for range.	You must provide a later date and time than *first*.	"Frequency Options" on page A-26
seqno	Sequence number of a specific error.	You must provide the number of an error in the error table.	Integer

Table A-4. Options for the cdr error command

Long Form	Short Form	Meaning
(no options specified)		Print the current list of errors and then mark them as reviewed. Enterprise Replication does not display errors marked as reviewed.
--all	**-a**	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

Run the **cdr error** command to examine replication errors. Sometimes a command succeeds on the server on which it is run but fails on one of the remote servers. For example, if you run the **cdr define replicate** command on **server1**, but the table name is misspelled on **server2**, the command succeeds on **server1** and seems to complete successfully. You can use **cdr error -c server2** to see why replication is failing.

The **cdr error** command also allows you to administer the **syscdrerror** table remotely. The **syscdrerror** table on each replication server contains errors for all replication servers, unless the replication server is a leaf node. The **syscdrerror** tables on leaf nodes do not contain errors for other replication servers. The reviewed flag indicates which errors are new errors while keeping the old errors in the table. For example, you can run **cdr error** periodically and append the output to a file.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following command shows the current list of errors on database server **hill**:

```
cdr error --connect=hill
```

After the errors are shown, 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, 2008:

```
cdr error -p "2008-05-01 14:56:00"
```

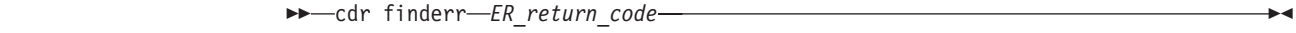
The following command deletes all errors from the error table that occurred at or after noon on May 1, 2008 and before or at 2:56 in the afternoon on May 1, 2008:

```
cdr error -p "2008-05-01 14:56:00,2008-05-01 12:00:00"
```

cdr finderr

The **cdr finderr** command looks up a specific Enterprise Replication return code and displays the corresponding error text.

Syntax



Element	Purpose	Restrictions
ER_return_code	Enterprise Replication return code to look up.	Must be a positive integer.

You can also view the Enterprise Replication return codes in the file **$INFORMIXDIR/incl/esql/cdrerr.h**.

You can run this command from within an SQL statement by using the SQL administration API.

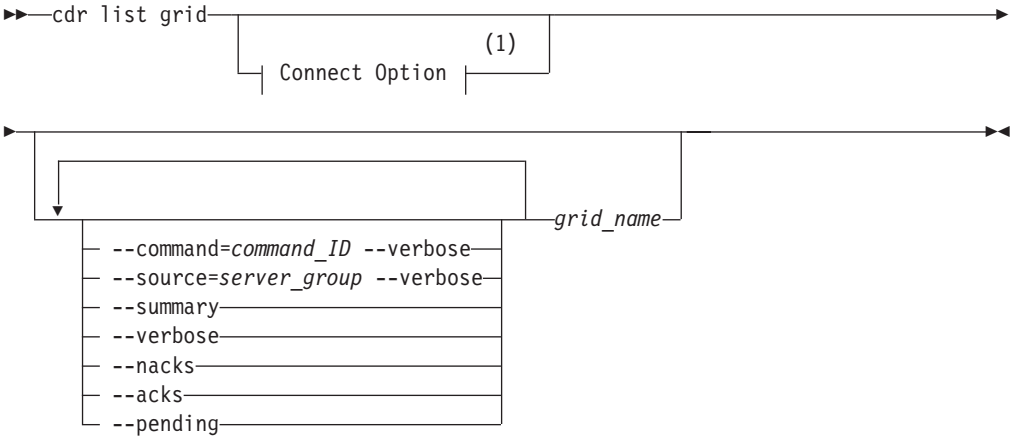
Related concepts:

"Return Codes for the cdr Utility" on page A-8

cdr list grid

The **cdr list grid** command shows information about a grid.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
command_ID	The ID of a specific command that was run from the grid.		An integer.
grid_name	Name of the grid.	Must be the name of an existing grid.	"Long Identifiers" on page A-3
server_group	Name of a database server group from which the command was run.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3

The following table describes the **cdr list grid** options.

Long Form	Short Form	Meaning
--acks	**-a**	Displays the servers in the grid and the commands that succeeded on one or more servers.
--command=	**-C**	Displays the servers in the grid and the specified command.
--nacks	**-n**	Displays the servers in the grid and the commands that failed on one or more servers.

Long Form	Short Form	Meaning
--pending	**-p**	Displays the servers in the grid and the commands that are in progress. A command can be pending because the transaction has not completed processing on the target server, the target server is down, or the target server was added to the grid after the command was run.
--source=	**-S**	Displays the servers in the grid and the commands that were run from the specified server.
--summary	**-s**	Displays the servers in the grid and the commands that were run on the grid.
--verbose	**-v**	Displays the servers in the grid, the commands that were run on the grid, and the results of the commands on each server in the grid.

Usage

Use the **cdr list grid** command to view information about servers in the grid, and about the commands that were run on servers in the grid.

If you run the **cdr list grid** command without any options or without a grid name, the output shows the list of grids.

Servers in the grid on which users are authorized to run grid commands are marked with an asterisk (*).

When you add a server to the grid, any commands that were previously run through the grid have a status of PENDING for that server. If you want to run previous grid commands on a new grid server, use the **ifx_grid_redo()** procedure. If you do not want to run previous grid commands on a new server, you can purge the commands by running the **ifx_grid_purge()** procedure.

When you run an SQL administration API command, the status of the grid command does not necessarily reflect whether the SQL administration API command succeeded. The grid command can have a status of ACK even if the SQL administration API command failed. The **cdr list grid** command shows the return codes of the SQL administration API commands. The **task()** function returns a message indicating whether the command succeeded. The **admin()** function returns an integer which if it is a positive number indicates that the command succeeded.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 220, 222.

For information on these error codes, see “Return Codes for the cdr Utility” on page A-8.

Examples

The examples in this section show the output of the **cdr list grid** command on a grid **grid1** that contains three servers: **cdr1**, **cdr2**, and **cdr3**.

Example 1: Display grid members

The following command displays the members of the **grid1** grid:

```
cdr list grid grid1
```

The output of the previous command is:

```
Grid              Node              User
----------------- ----------------- ----------------
grid1             cdr1*             bill
                  cdr2
                  cdr3
```

This output shows that the grid contains three member servers and that the authorized user **bill** can run grid routines from the server **cdr1**.

Example 2: Display verbose information about commands

The following command displays verbose information about a series of commands and their results on each server in the grid:

```
cdr list grid --verbose grid1
```

The output of the previous command is:

```
Grid              Node              User
----------------- ----------------- ----------------
grid1             cdr1*             bill
                  cdr2
                  cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create database tstdb with log
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3

Node:cdr1 Stmtid:2 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create table tab1 (col1 int, col2 int)
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3

Node:cdr1 Stmtid:3 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create procedure load(maxnum int)
define tnum int;
for tnum = 1 to maxnum
    insert into tab1 values (tnum, 1);
end for;
end procedure;
ACK cdr1 2010-05-27 15:21:57
ACK cdr2 2010-05-27 15:21:58
PENDING cdr3
```

This output shows each command and that all commands succeeded on servers **cdr1** and **cdr2** but are pending on the **cdr3** server because it is offline.

Example 3: Display errors

In this example, the **cdr3** server already has a database with the same name as the database in the CREATE DATABASE statement: therefore, the CREATE DATABASE and CREATE TABLE statements fail. The following command displays information about commands run within the grid that resulted in an error:

```
cdr list grid --nacks grid1
```

The output of the previous command is:

```
Grid              Node              User
----------------- ----------------- ----------------
grid1             cdr1*             bill
                  cdr2
                  cdr3
Details for grid grid1

Node:cdr1 Stmtid:1 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create database tstdb with log
NACK cdr3 2010-05-27 15:39:21 SQLERR:-330 ISAMERR:-100

Node:cdr1 Stmtid:2 User:dba1 Database:tstdb 2010-05-27 15:21:57
Tag:test
create table tab1 (col1 int, col2 int)
NACK cdr3 2010-05-27 15:39:21 SQLERR:-310 ISAMERR:0
     Grid Apply Transaction Failure
```

This output shows the SQL and ISAM error codes associated with the failed statements.

Related concepts:

"Example of setting up a replication system with a grid" on page 7-2

Related tasks:

"Maintaining the grid" on page 7-5

"Rerunning failed grid routines" on page 7-13

Related reference:

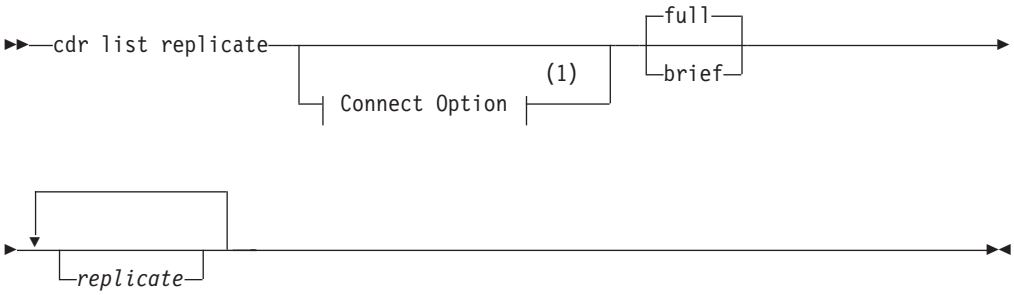
"cdr define grid" on page A-58

"cdr change grid" on page A-30

cdr list replicate

The **cdr list replicate** command displays information about the replicates on the current server.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
replicate	Name of the replicates.	The replicates must exist.	"Long Identifiers" on page A-3

Usage

The **cdr list replicate** command displays information about replicates (the **full** option). 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.

To display only replicate names and participant information, use the **brief** option.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list replicate** is executed on a leaf server, it displays incomplete information about the other database servers.

The **cdr list replicate** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list replicate** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

Output Description

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 acknowledgment 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
`Deletewins`	Specifies that the replicate uses the delete wins conflict-resolution rule
`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 OPTIONS field can include the following values.

Value	Description
ats	Indicates that ATS files are generated if transactions fail to be applied at the target server.
firetrigger	Indicates that the rows that this replicate inserts fire triggers at the destination.
floatcanon	Indicates that floating-point numbers are replicated in machine-independent decimal representation.
floatieee	Indicates that floating-point numbers are replicated in either 32-bit (for SMALLFLOAT) or 64-bit (for FLOAT) IEEE floating-point format.
fullrow	Indicates to replicate only changed columns and disable upserts.
ignoredel	Indicates that rows are retained if they are deleted on other nodes in the domain.
pendingsync	Indicates that the replication server has been enabled with the **cdr enable server** command but that the participant has not yet been synchronized with the rest of the domain. ATS and RIS files for this participant are not created if transactions are aborted.
ris	Indicates that RIS files are generated if transactions fail to be applied at the target server.
row	Indicates that the replicate uses row scope.
transaction	Indicates that the replicate uses transaction scope.
UTF8	Indicates that code set conversion between replicates is enabled.

The REPLTYPE field can include the following values. If the REPLTYPE is not shown, the replicate is a classic replicate (neither a master or a shadow replicate).

Value	Description
Master	Indicates that the replicate is defined as a master replicate.
Shadow	Indicates that the replicate is a shadow replicate. A shadow replicate can also be a master replicate.
Grid	Indicates that the replicate belongs to a grid replicate set.

The PARENT REPLICATE field appears only for shadow replicates. It shows the name of the replicate on which the shadow replicate is based.

Examples

The following example displays a list of the replicates on the current server with full details:

```
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
REPLTYPE:         Master

REPLICATE:        Repl2
STATE:            Inactive
CONFLICT:         Deletewins
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      bank:joe.account
OPTIONS:          row,ris,ats
REPLTYPE:         Master,Shadow
PARENT REPLICATE: Repl1
```

If the replicate belongs to a grid replicate set, the REPLTYPE field includes Grid.

```
CURRENTLY DEFINED REPLICATES
-------------------------------------------
REPLICATE:        grid_6553604_100_3
STATE:            Active ON:g_delhi
CONFLICT:         Always Apply
FREQUENCY:        immediate
QUEUE SIZE:       0
PARTICIPANT:      tdb:nagaraju.t1
OPTIONS:          row,ris,fullrow
REPLID:           6553605 / 0x640005
REPLMODE:         PRIMARY  ON:g_delhi
APPLY-AS:         INFORMIX ON:g_delhi
REPLTYPE:         Master,Grid
```

The PARENT REPLICATE field only appears if the replicate is a shadow replicate.

The following example displays a list of the replicates on the current server with brief details:

```
cdr list replicate brief
```

The output from the previous command might be the following:

```
REPLICATE   TABLE                           SELECT
------------------------------------------------------------------
Repl1       bank@g_newyork:joe.teller       select * from joe.teller
Repl1       bank@g_sanfrancisco:joe.teller  select * from joe.teller
Repl2       bank@g_portland:joe.teller      select * from joe.teller
Repl2       bank@g_atlanta:joe.teller       select * from joe.teller
```

The following example specifies the names of replicate:

```
cdr list repl brief 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
```

Related tasks:

"Adding an existing replicate to a grid replicate set" on page 7-7

"Viewing grid information" on page 7-7

Related reference:

"cdr change replicate" on page A-32

"cdr define replicate" on page A-61

"cdr delete replicate" on page A-79

"cdr modify replicate" on page A-108

"cdr resume replicate" on page A-126

"cdr start replicate" on page A-131

"cdr stop replicate" on page A-152

"cdr suspend replicate" on page A-155

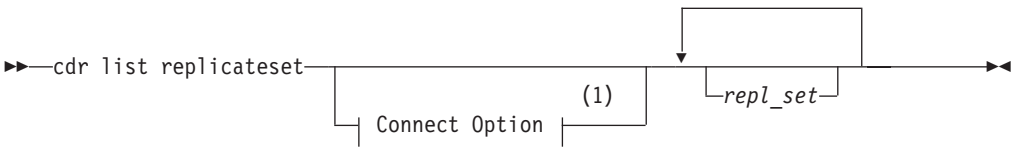
"cdr swap shadow" on page A-159

"cdr list replicateset"

cdr list replicateset

The **cdr list replicateset** command displays information about the replication sets defined on the current server.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_set	Name of the replicate set.	The replicate set must exist.	"Long Identifiers" on page A-3

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*.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list replicateset** is executed against a leaf server, it displays incomplete information about the other database servers.

If you specify the name of a grid replicate set, the command displays the names of the replicates that were automatically created through the grid and any replicates manually added to the grid replicate set. The name of the grid replicate set is the same as the name of the grid.

The **cdr list replicateset** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list replicateset** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

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:

```
Ex T REPLSET                  PARTICIPANTS
-----------------------------------------------
N  Y  g1                        Repl1, Repl4
N  Y  g2                        Repl2, Repl3, Repl5
```

The Ex field shows whether the replicate set is exclusive. The T field shows whether the replicate set was created from a template.

This example displays information for all the replicates in the replicate set **g1**:

```
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
REPLTYPE:      Master

REPLICATE:     Repl4
STATE:         Inactive
CONFLICT:      Deletewins
FREQUENCY:     immediate
QUEUE SIZE:    0
PARTICIPANT:   bank:arthur.teller
OPTIONS:       row,ris,ats
REPLTYPE:      Master
```

The information supplied for each replicate is the same as the information provided by the **cdr list replicate** command.

Related concepts:

"Example of setting up a replication system with a grid" on page 7-2

Related tasks:

"Viewing grid information" on page 7-7

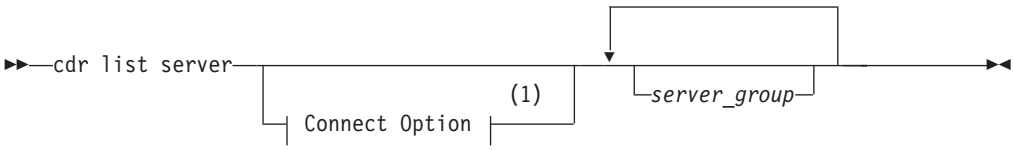
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr delete replicateset" on page A-80
"cdr define replicate" on page A-61
"cdr modify replicateset" on page A-111
"cdr resume replicateset" on page A-127
"cdr start replicateset" on page A-134
"cdr stop replicateset" on page A-154
"cdr suspend replicateset" on page A-157
"cdr list replicate" on page A-97

cdr list server

The **cdr list server** command displays a list of the Enterprise Replication servers that are visible to the server on which the command is run.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
server_group	Name of the server group.	The database server groups must be defined for Enterprise Replication.	

Usage

The **cdr list server** command displays information about servers. You do not need to be user **informix** to use this command; any user can run it.

The **cdr list server** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list server** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the **onspaces** utility.

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.

In hierarchical topology, leaf servers only have information about their parent database servers in the Enterprise Replication domain. Therefore, when **cdr list server** is executed against a leaf server, it displays incomplete information about the other database servers.

Output Description

The SERVER and ID columns display the name and unique identifier of the Enterprise Replication server group.

The STATE column can have the following values.

Active	The server is active and replicating data.
Deleted	The server has been deleted; it is not capturing or delivering data and the queues are being drained.
Disabled	The server is disabled. It is not capturing or delivering data, but its delete tables are being maintained.
Quiescent	The server is in the process of being defined.
Suspended	Delivery of replication data to the server is suspended.

The STATUS column can have the following values.

Connected	The connection is active.
Connecting	The connection is being established.
Disconnect	The connection was explicitly disconnected.
Disconnected will attempt reconnect	The connection was disconnected but is being reattempted.
Dropped	The connection was disconnected due to a network error because the server is unavailable.
Error	The connection was disconnected due to an error (check the log and contact customer support, if necessary).
Failed	The connection attempt failed.
Local	Identifies that this server is the local server as opposed to a remote server.
Timeout	The connection attempt has timed out, but will be reattempted.

The QUEUE column displays the size of the queue for the server group.

The CONNECTION CHANGED column displays the most recent time that the status of the server connection was changed.

Examples

In the following examples, **usa**, **italy**, and **france** are root servers, **denver** is a nonroot server, and **miami** is a leaf server. The **usa** server is the parent of **denver**, and **denver** is the parent of **miami**.

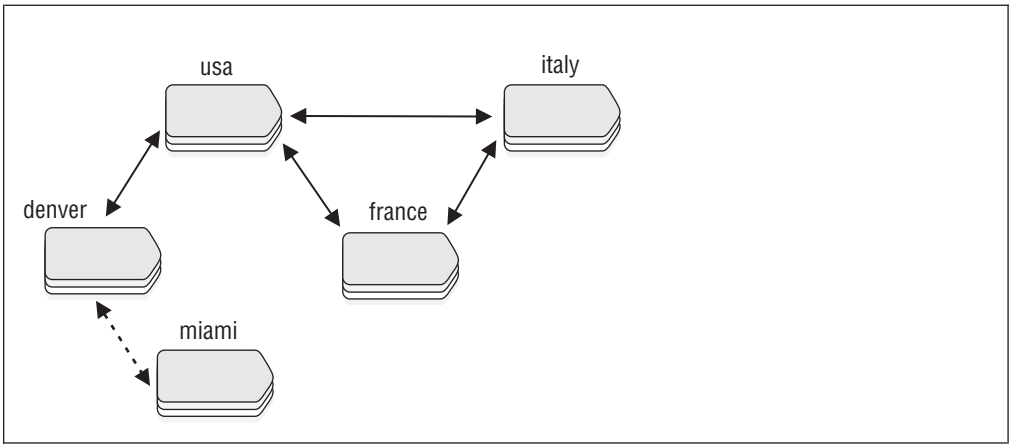


Figure A-2. cdr list server example

When the **cdr list server** command includes the name of a database server group, the output displays the attributes of that database server. The following commands and example output illustrate how the **cdr list server** command displays server information.

In this example, the server **g_usa** generates ATS and RIS files in XML format, has an idle time out of 15 seconds, and is a hub server.

```
cdr list server g_usa

NAME      ID     ATTRIBUTES
--------------------------------------------------
g_usa     1      atsrisformat=xml timeout=15 hub
```

In this example, the **g_denver** server shows the **g_usa** server as its root server.

```
cdr list server -c denver g_denver
NAME      ID     ATTRIBUTES
--------------------------------------------------
g_denver  27     root=g_usa
```

In this example, the attributes of the **g_denver** server are shown from the perspective of the **italy** server. The **g_denver** server has the **g_usa** server as its root server and uses the **g_usa** server to forward replicated transactions between it and the **italy** server.

```
cdr list server -c italy g_denver

NAME      ID     ATTRIBUTES
--------------------------------------------------
g_denver  27     root=g_usa forward=g_usa
```

In this example, the **g_miami** server shows the **g_denver** server as its root server and that it is a leaf server.

```
cdr list server g_miami

NAME      ID     ATTRIBUTES
--------------------------------------------------
g_miami   4      root=g_denver leaf
```

The following example shows possible output for the **cdr list server** command if no server groups are specified:

```
cdr list server
SERVER         ID STATE    STATUS     QUEUE CONNECTION CHANGED
-------------------------------------------------------------
```

```
g_denver        1 Active   Local      0
g_miami         2 Active   Connected  0     Mar 19 13:48:44
g_usa           3 Active   Connected  0     Mar 19 13:48:40
g_france        4 Active   Connected  0     Mar 19 13:48:41
g_italy         5 Active   Connected  0     Mar 19 13:48:45
```

Related reference:

"cdr connect server" on page A-57

"cdr define server" on page A-71

"cdr delete server" on page A-81

"cdr disconnect server" on page A-88

"cdr modify server" on page A-112

"cdr resume server" on page A-128

"cdr start" on page A-129

"cdr suspend server" on page A-158

"cdr view" on page A-169

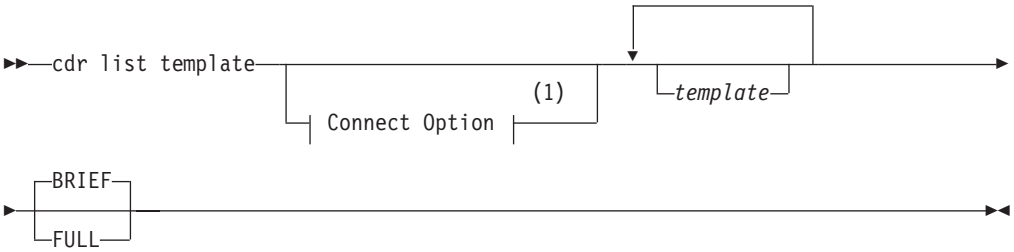
"cdr resume replicate" on page A-126

"cdr resume replicateset" on page A-127

cdr list template

The **cdr list template** command displays information about the templates on the server on which the command is run.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
template	Name of the template.	The template must exist.	"Long Identifiers" on page A-3

Usage

The **cdr list template** command displays information about templates. If no templates are named, the command lists all templates in the Enterprise Replication domain. If one or more templates are named, the command displays the names, database names, and table names for those templates.

To display detailed information for your templates, use the **FULL** option.

You do not need to be user **informix** to use this command; any user can run it.

In hierarchical topology, leaf servers have limited information about other database servers in the Enterprise Replication domain. Therefore, when **cdr list template** is executed against a leaf server, it displays incomplete information about the other database servers.

The **cdr list template** command can be used while the replication server is in DDRBLOCK mode. Before using the **cdr list template** command you must set the DBSPACETEMP configuration parameter and create a temporary dbspace with the onspaces utility.

Examples

The following example displays detailed information about the templates on the current server:

```
cdr list template
```

The output from the previous command might be the following:

```
TEMPLATE            DATABASE            TABLES
============================================
tem1                newcars             table1
                    newcars             table2
                    newcars             table3
tem2                carparts            table1
                    carparts            table3
```

The following example displays detailed information about the template **tem1**:

```
cdr list template tem1
```

The output from the previous command might be the following:

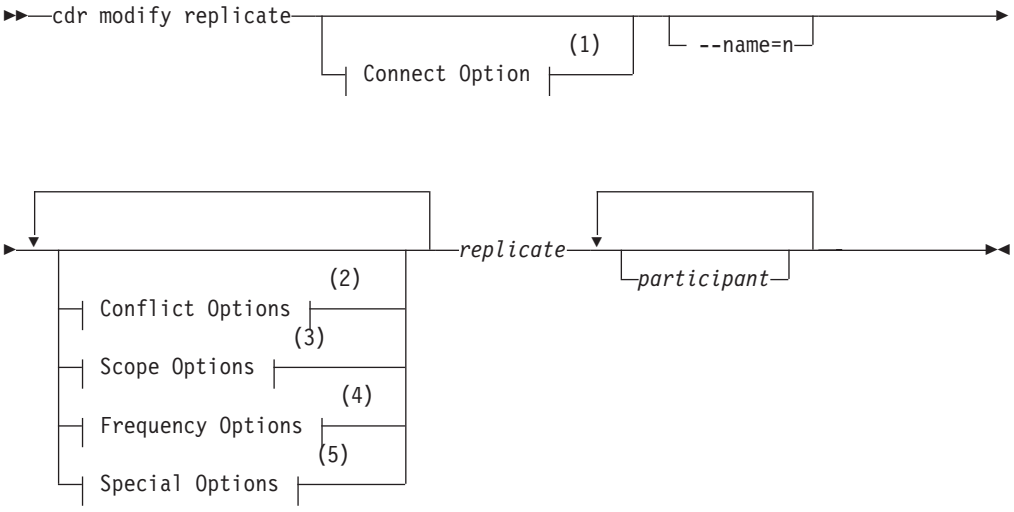
```
CURRENTLY DEFINED TEMPLATES
===========================
TEMPLATE:    tem1
TEMPLATE ID: 6553605
SERVER:      utah
DATABASE:    newcars
REPLICATE:   tem1_utah_2_1_table1
OWNER:       pravin
TABLE:       table1

TEMPLATE:    tem1
TEMPLATE ID: 6553605
SERVER:      utah
DATABASE:    newcars
REPLICATE:   tem1_utah_2_2_table2
OWNER:       pravin
TABLE:       table2

TEMPLATE:    tem1
TEMPLATE ID: 6553605
SERVER:      utah
DATABASE:    newcars
REPLICATE:   tem1_utah_2_3_table3
OWNER:       pravin
TABLE:       table3
```

Related reference:
"cdr define template" on page A-75
"cdr realize template" on page A-114

cdr modify replicate

The **cdr modify replicate** command modifies replicate attributes.

Syntax



Notes:

1 See "Connect Option" on page A-3.

2 See "Conflict Options" on page A-63.

3 See "Scope Options" on page A-65.

4 See "Frequency Options" on page A-26.

5 See "Special Options" on page A-109.

Element	Purpose	Restrictions	Syntax
participant	Name of a participant in the replication.	The participant must be a member of the replicate.	"Participant and participant modifier" on page A-4
replicate	Name of the replicate to modify.	The replicate name must exist.	"Long Identifiers" on page A-3

The following table describes the option to **cdr modify replicate**.

Long Form	Short Form	Meaning
--name=n	**-n n**	Removes the name verification attribute from a master replicate.

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.

To add or delete a participant, use the **cdr change replicate** command.

If you change the conflict resolution rule with **cdr modify replicate**, you must also specify the scope with the**---scope** option, even if you are not changing the scope.

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.

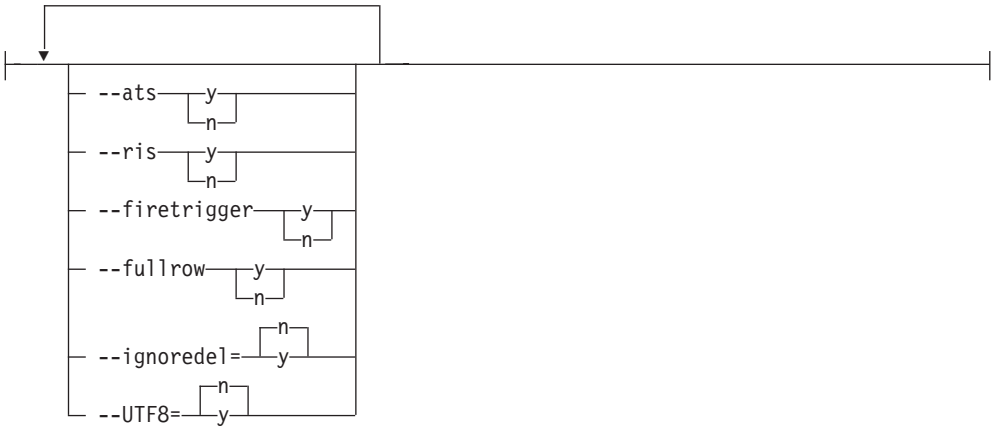
  However, you can change from time stamp resolution to SPL routine resolution or from SPL routine resolution to time stamp.
- The **--ats**, **--ris**, --**firetrigger**, and **--fullrow** options require a yes (**y**) or no (**n**) argument.

When you run the **cdr modify replicate** command, an event alarm with a class ID of 63 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Special Options

Special Options:



The following table describes the special options to **cdr modify replicate**.

Long Form	Short Form	Meaning
--ats y or **--ats n**	**-A y** or **-A n**	Activates (**y**) or deactivates (**n**) aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
--firetrigger y or **--firetrigger n**	**-T y** or **-T n**	Causes the rows inserted by this replicate to fire (**y**) or not fire (**n**) triggers at the destination.
--fullrow y or **--fullrow n**	**-f y** or **-f n**	Specifies to (**y**) replicate the full row and enable upserts or (**n**) replicate only changed columns and disable upserts.
--ignoredel=	**-D**	Specifies whether to retain deleted rows on other nodes: • y: Indicates that rows are retained if they are deleted on other nodes in the Enterprise Replication domain. You cannot use this option if you specify **deletewins** as the conflict resolution rule; the **cdr define replicate** command fails when these contradictory options are combined. • n: (Default) Indicates that deleted rows are deleted on all nodes in the Enterprise Replication domain.
--ris y or **--ris n**	**-R y** or **-R n**	Activates (**y**) or deactivates (**n**) row-information spooling for replicate row data that fails conflict resolution or encounters replication-order problems.
--UTF8=y or **--UTF8=n**	None	Specifies whether to enable conversion to and from UTF-8 (Unicode) when replicating data between servers that use different code sets. If **–UTF8=y** is specified, then character columns are converted to UTF-8 when the row is copied into the transmission queue. When the replicated row is applied on the target server, the data is converted from UTF-8 to the code set used on the target server. No attempt is made to convert character data contained within opaque data types such as user-defined data types or DataBlade data. If **–UTF8=n** is specified, then code set conversion is ignored.

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"
```

Related concepts:

"Frequency Options" on page A-26

Related tasks:

"Enabling ATS and RIS File Generation" on page 9-4

"Creating Strict Master Replicates" on page 6-9

Related reference:

"cdr change replicate" on page A-32

"cdr define replicate" on page A-61

"cdr delete replicate" on page A-79

"cdr list replicate" on page A-97

"cdr resume replicate" on page A-126

"cdr start replicate" on page A-131

"cdr stop replicate" on page A-152

"cdr suspend replicate" on page A-155

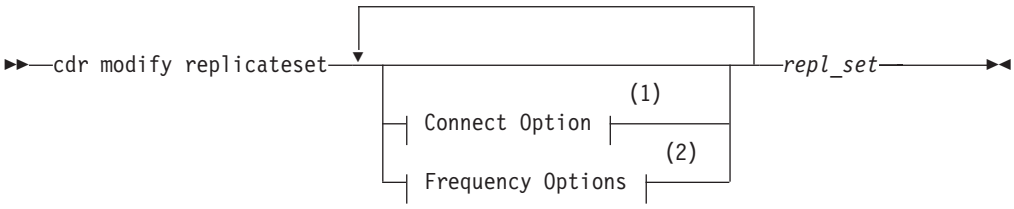
"Enterprise Replication Event Alarms" on page 9-21

"Participant and participant modifier" on page A-4

cdr modify replicateset

The **cdr modify replicateset** command modifies all the replicates in a replicate set.

Syntax



Notes:

1 See "Connect Option" on page A-3.

2 See "Frequency Options" on page A-26.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to modify.	The replicate set must exist.	"Long Identifiers" on page A-3

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.

You cannot change whether a replicate set is exclusive or not.

When you run the **cdr modify replicateset** command, an event alarm with a class ID of 64 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server (the server specified by the **INFORMIXSERVER** environment variable) and modifies the replicate set **sales_set** to process replication data every hour:

```
cdr mod replset --every=60 sales_set
```

Related concepts:

"Frequency Options" on page A-26

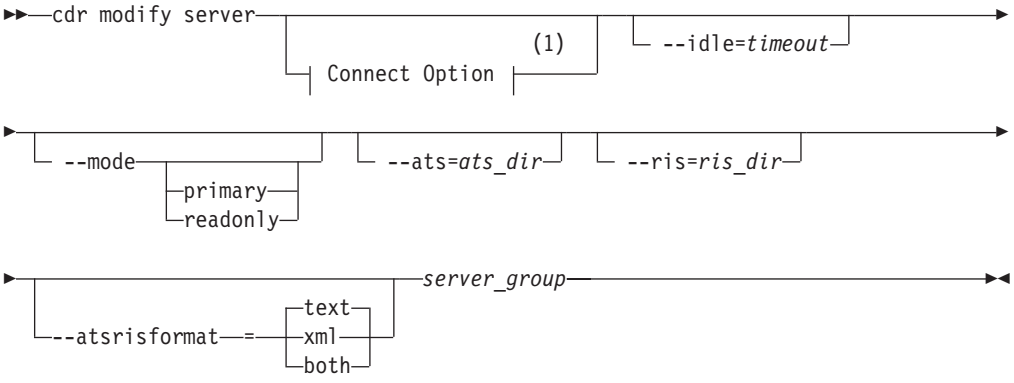
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr delete replicateset" on page A-80
"cdr list replicateset" on page A-101
"cdr define replicate" on page A-61
"cdr resume replicateset" on page A-127
"cdr start replicateset" on page A-134
"cdr stop replicateset" on page A-154
"cdr suspend replicateset" on page A-157
"Enterprise Replication Event Alarms" on page 9-21

cdr modify server

The **cdr modify server** command modifies the Enterprise Replication attributes of a database server.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
server_group	Name of a database server group to modify.	The database server group must be defined in Enterprise Replication.	
timeout	Idle timeout for this server.	Must be an integer number of minutes. 0 indicates no timeout. The maximum value is 32,767.	Integer.
ats_dir	Name of Aborted Transaction Spooling directory.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of **/dev/null** (UNIX) or **NUL** (Windows) prevents ATS file generation.	Follows naming conventions on your operating system.
ris_dir	Name of the Row Information Spooling directory.	Must be a full path name. The path for the directory can be no longer than 256 bytes. A value of **/dev/null** (UNIX) or **NUL** (Windows) prevents RIS file generation.	Follows naming conventions on your operating system.

The following table describes the options to **cdr modify server**.

Long Form	Short Form	Meaning
--ats=	**-A**	Activates aborted-transaction spooling for replicate transactions that fail to be applied to the target database.
–atsrisformat=	**-X**	Specifies the format of ATS and RIS files: • **text** indicates that ATS and RIS files are generated in standard text format. • **xml** indicates that ATS and RIS files are generated in XML format. • **both** indicates that ATS and RIS files are generated in both standard text format and XML format.
--idle=	**-i**	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.
--mode	**-m**	Changes the mode of all replicates using this server to either primary (**primary**) or to read-only (**readonly**). **Note:** The **-m** option only affects replicates whose conflict resolution is ignore.
--ris=	**-R**	Activates row-information spooling for replicate-row data that fails conflict resolution or encounters replication-order problems.

Usage

The **cdr modify server** command modifies the replication server *server_group*.

When you run the **cdr modify server** command, an event alarm with a class ID of 70 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

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 are generated into the directory **/cdr/atsdir** in both text and XML format.

```
cdr modify server -c paris -i 10 -A /cdr/atsdir \
-X both 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 p g_geometrix
```

Related concepts:

Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1

Related tasks:

"Enabling ATS and RIS File Generation" on page 9-4

"Disabling ATS and RIS File Generation" on page 9-13

"Customizing the Replication Server Definition" on page 6-6

Related reference:

"cdr connect server" on page A-57

"cdr define server" on page A-71

"cdr delete server" on page A-81

"cdr disconnect server" on page A-88

"cdr list server" on page A-103

"cdr resume server" on page A-128

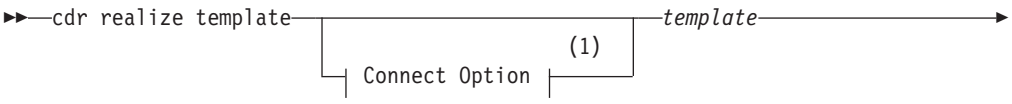
"cdr suspend server" on page A-158

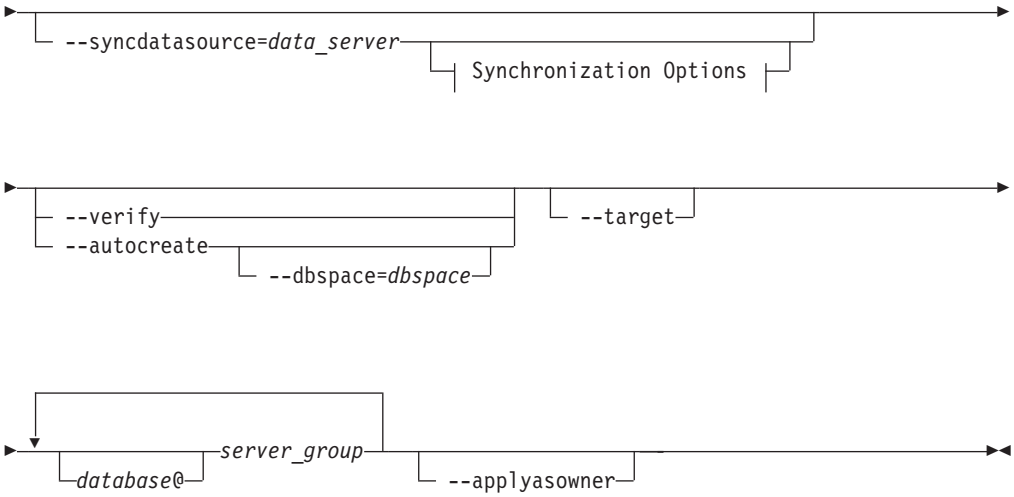
"Enterprise Replication Event Alarms" on page 9-21

cdr realize template

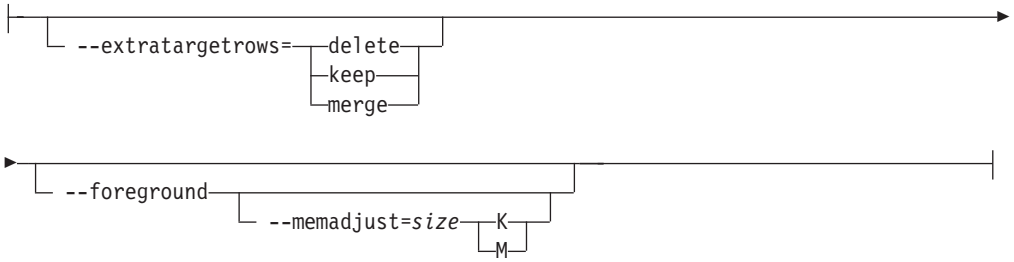
The **cdr realize template** command creates the replicates, replicate set, and participant tables as specified in a template, and then synchronizes data on all or a subset of the database servers within the replication domain.

Syntax





Synchronization Options:



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
database	Name of the database that includes the table to be replicated.	The database server must be in an Enterprise Replication domain.	"Long Identifiers" on page A-3
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be in an Enterprise Replication domain.	
dbspace	The name of the dbspace for Enterprise Replication to use when creating tables.	The dbspace must exist on all the database servers listed. If you do not specify a dbspace name and new tables are created, they are created in the default dbspace.	
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-3

Element	Purpose	Restrictions	Syntax
*size***K** or *size* **M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	
template	The name of the template.	The template must exist. Use the **cdr define template** command to create the template.	"Long Identifiers" on page A-3

The following table describes the special options to **cdr realize template**.

Long Form	Short Form	Meaning
--applyasowner	**-o**	Specifies that any tables created when you realize the template are owned by the owner of the source tables. By default, the tables are owned by the user **informix**.
--autocreate	**-u**	Specifies that if the tables in the template definition do not exist in the databases on the target servers, they are created automatically. However, the tables cannot contain columns with user-defined data types. **Note:** Tables created with autocreate do not automatically include non-primary key indices, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with autocreate require the use of these objects you must manually create them.
--dbspace=	**-D**	Specifies the dbspace in which the automatically created objects are placed. If not specified, then the default dbspace is used.
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.

Long Form	Short Form	Meaning
--syncdatasource=	**-S**	Specifies which server is the source of the data that is used to synchronize all the other servers listed in the **cdr realize template** command. The server listed with this option must either be listed as one of the servers on which to realize the template, or it must already have the template.
--target	**-t**	Specifies that all of the servers listed in the command become receive-only servers, including the source server, unless the template has already been realized on the source server. If you use this option, you must run the **cdr realize template** command twice: once to realize the template on the source server and other primary servers, and again to realize the template on receive-only servers.
--verify	**-v**	Specifies that the **cdr realize template** command verifies that the database, tables, column data types are correct on all listed servers, but does not realize the template.

Usage

Before you can use the **cdr realize template** command, you must define Enterprise Replication servers using the **cdr define server** command and define the template using the **cdr define template** command. You should also create the database to be replicated on all database servers in the replication domain. However, only the database on the synchronization data source server needs to be populated with data.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The **cdr realize template** command performs the following tasks:

- If you specify the **--autocreate** option, creates database tables on the target servers.

  Recommendation: If you use **--autocreate**, specify a dbspace name. If you do not specify a dbspace name, tables are created in the root dbspace, which is not recommended. Also note that tables created with autocreate do not automatically include non-primary key indices, defaults, constraints (including foreign constraints), triggers, or permissions. If the tables you create with autocreate require the use of these objects you must manually create them.
- If you specify the **--verify** option, verifies the database, tables, column data types, and primary keys on all participating servers; however, the template is not realized.
- If you specify the **--syncdatasource** option, synchronizes the data from the source database with the databases specified by this command. If you specify the **--foreground** option, runs synchronization as a foreground process. If you specify the **--memadjust** option, increases the size of the send queue from the value of the CDR_QUEUEMEM configuration parameter.

If you are running this command with the **--syncdatasource** option as a DBSA, you need to have certain permissions granted to you on several system tables in the **syscdr** database. For more information, see "Preparing for Role Separation (UNIX)" on page 4-22.

- Verifies the database and table attributes to ensure that proper replication can be performed on each database.
- Creates replicates as master replicates.
- Creates a replicate set for the new replicates.
- Starts the replicates.

The replicates and replicate set created from a template have generated names. Use the **cdr list template** command to see the names of the replicates and replicate set associated with a particular template.

You can run this command from within an SQL statement by using the SQL administration API.

You can run the **cdr check queue --qname=cntrlq** command to wait for the **cdr realize template** command to be applied at all Enterprise Replication servers before running the data synchronization task.

Examples

The following example illustrates the **cdr realize template** command:

```
cdr realize template tem1 -c detroit\
new_cars@detroit new_cars0@chicago new_cars1@newark\
new_cars2@columbus
```

Line 1 specifies that the template name is **tem1** and the server to connect to is the **detroit** server. Lines 2 and 3 list the names of the databases and database servers on which to realize the template.

The following example illustrates realizing the template on the source server, and then, creating the databases and tables, and loading data on the target database servers:

```
cdr realize template tem1 -c detroit\
--syncdatasource=detroit --extratargetrows=keep\
--foreground --memadjust=50M\
--target chicago newark columbus
```

Line 1 realizes the template on the **detroit** server, as a primary server by default.

Line 2 specifies to use the **detroit** server as the source of the data to replicate to all other participating servers. If Enterprise Replication encounters any rows on the **chicago**, **newark**, or **columbus** servers that do not exist on the **detroit** server, those rows are kept.

Line 3 specifies that the synchronization operation is done in the foreground, and the size of the send queue is set to 50 MB.

Line 4 specifies the participant type for each server. The **--target** option makes all servers receive-only participants.

The following example verifies the database and table attributes on the **chicago**, **newark**, and **columbus** servers; the template is not realized on these servers:

```
cdr realize template tem1 -c detroit\
--verify chicago newark columbus
```

Related concepts:

"Database Server Groups" on page 4-2

"Preparing for Role Separation (UNIX)" on page 4-22

Related reference:

"cdr define template" on page A-75

"cdr delete template" on page A-84

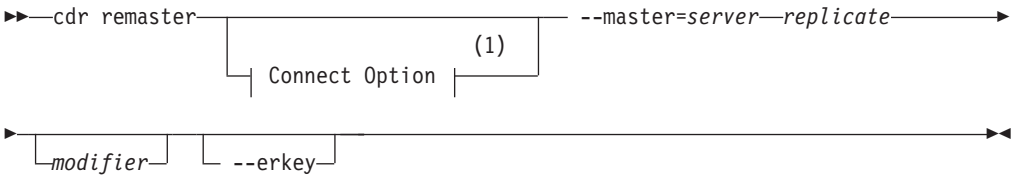
"cdr list template" on page A-106

"cdr define server" on page A-71

cdr remaster

The **cdr remaster** command changes the SELECT clause or the server from which to base the master replicate definition of an existing master replicate. This command can also convert a classic (non-master) replicate to a master replicate.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
modifier	Specifies the rows and columns to replicate.		"Participant and participant modifier" on page A-4
replicate	Name of the replicate to be mastered.	The replicate must exist.	"Long Identifiers" on page A-3
server	Name of the database server group from which to base the master replicate definition.	The name must be the database server group name.	"Long Identifiers" on page A-3

The following table describes the option to **cdr remaster**.

Long Form	Short Form	Meaning
--erkey	**-K**	Includes the ERKEY shadow columns, **ifx_erkey_1**, **ifx_erkey_2**, and **ifx_erkey_3**, in the participant definition, if the table being replicated has the ERKEY shadow columns. The ERKEY shadow columns are used in place of a primary key.

Long Form	Short Form	Meaning
--master=	**-M**	Specifies that the replicate being created is a master replicate.

Usage

Use the **cdr remaster** command to perform one of the following tasks:

- Convert a classic replicate to a master replicate. Master replicates ensure schema consistency among the participants in the replicates.
- Update the definition of a master replicate whose participant was changed in an alter operation. You can change the SELECT clause or the server from which to base the master replicate definition.

To use the **cdr remaster** command, the master replicate definition must be created with name verification turned on, by using the **cdr define replicate** command with the **--name=y** option.

Use the **--erkey** option if you are adding ERKEY columns to the participant definition, or if you are changing a participant definition that contains the ERKEY shadow columns.

You can run this command from within an SQL statement by using the SQL administration API.

As part of its processing, the **cdr remaster** command creates a shadow replicate. The shadow replicate is named as follows:

```
Shadow_4_basereplicatename_GMTtime_GIDlocalCDRID_PIDpid
```

The shadow replicate name is composed of the following parts:

- The *basereplicatename* is the name of the replicate being remastered (up to 64 characters of this name are used).
- The *localCDRID* is the CDR group ID of the server you specified with the **--connect** option. You can obtain this ID using the **onstat -g cat servers** command or by looking in the SQLHOSTS file. Alternatively, you can query the **syscdrserver** view in the **sysmaster** database.
- The *pid* is the process ID of the client computer.

An example of a shadow replicate name is:

```
Shadow_4_Repl1_GMT1090373046_GID10_PID28836
```

Examples

The following example shows the original definition of the master replicate before the alter operation:

```
cdr define repl --master=delhi -C timestamp\
newrepl "test@delhi.tab" "select col1, col2 from tab"\
```

This example shows the **cdr remaster** command adding a column, **col3**, in the **newrepl** participant:

```
cdr remaster --master=delhi newrepl\
"select col1, col2, col3 from tab"
```

This example shows adding the ERKEY shadow columns after the table was altered to include them:

```
cdr remaster --master=delhi newrepl --erkey\
"select col1, col2, col3 from tab"
```

This example shows changing the participant in the previous example to add another column and to continue to include the ERKEY shadow columns:

```
cdr remaster --master=delhi newrepl --erkey\
"select col1, col2, col3, col4 from tab"
```

Related tasks:

"Preparing tables without primary keys" on page 4-21

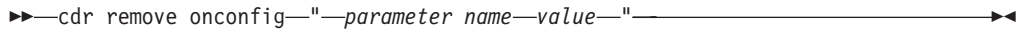
Related reference:

"cdr alter" on page A-29

cdr remove onconfig

The **cdr remove onconfig** command removes the specified value from a configuration parameter in the ONCONFIG file.

Syntax



Element	Purpose	Restrictions	Syntax
parameter name	The name of the configuration parameter from which to remove the value.	Not all configuration parameters can be changed with this command. Only the following parameters can be changed: • CDR_LOG_LAG_ACTION • CDR_LOG_STAGING_MAXSIZE • CDR_QDATA_SBSPACE • CDR_SUPRESS_ATSRISWARN • ENCRYPT_CIPHERS • ENCRYPT_MAC • ENCRYPT_MACFILE • CDR_ENV: – CDRSITES_731 – CDRSITES_92X – CDRSITES_10X	
value	The value of the configuration parameter to remove.	Must be an existing value of the configuration parameter.	Follows the syntax rules for the specific configuration parameter.

Usage

Use the **cdr remove onconfig** command to replace the existing value of an Enterprise Replication configuration parameter with a new value in the ONCONFIG file. You can set environment variables by using the CDR_ENV configuration parameter.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

Suppose the ENCRYPT_MAC configuration parameter is set to allow medium and high encryption levels, so that it appears in the ONCONFIG file as: `ENCRYPT_MAC medium,high`. The following command removes the medium encryption level and retains only the high encryption level:

```
cdr remove onconfig "ENCRYPT_MAC medium"
```

Suppose the CDR_SITES_92X environment variable specifies the cdrIDs of 3, 4, and 5, so that it appears in the ONCONFIG file as: CDR_ENV CDR_SITES_92X=3,4,5. The following command removes the cdrID of 3 from the list of supported version 9.2x servers:

```
cdr remove onconfig "CDR_ENV CDR_SITES_92X=3"
```

Related tasks:

"Dynamically Modifying Configuration Parameters for a Replication Server" on page 8-1

Related reference:

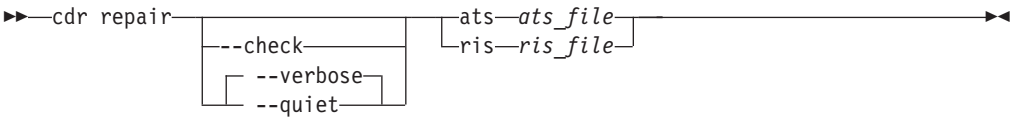
"cdr add onconfig" on page A-28

"cdr change onconfig" on page A-31

cdr repair

The **cdr repair** command synchronizes data based on ATS or RIS files.

Syntax



Element	Purpose	Restrictions	Syntax
ats_file	Name of the file for Aborted Transaction Spooling.	Must be a full path name and file name. The path for the directory can be no longer than 256 bytes. The file must be in text format; it cannot be in XML format.	Follows naming conventions on your operating system.
ris_file	Name of the file for Row Information Spooling.	Must be a full path name and file name. The path for the directory can be no longer than 256 characters. The file must be in text format; it cannot be in XML format.	Follows naming conventions on your operating system.

The following table describes the option to **cdr repair**.

Long Form	Short Form	Meaning
--check	**-C**	Check the consistency between the database server and the ATS or RIS file. Display repair operations to stderr, but do not perform the repair operations. In an active system, operations displayed with this option will not necessarily match those performed later during an actual repair.
--quiet	**-q**	Quiet mode. Repair operations are not displayed to stderr.
--verbose	**-v**	Verbose mode (default). All repair operations are displayed to stderr.

Usage

The **cdr repair** command reconciles rows that failed to be applied based on the information in the specified ATS or RIS file. If a row exists on the source database server, it is replicated again. If a row does not exist on the source database server, but does exist on the target server, then it is deleted from the target database server. By default, each of the repair operations is displayed to stderr.

If you are running this command as a DBSA, you must have read permission on the ATS and RIS files. Permissions on ATS and RIS files can be set with the **chown** operating system command.

The ATS or RIS file you specify in the **cdr repair** command must be in text format, which is the default format. You cannot specify the XML format of an ATS or RIS in the **cdr repair** command.

Before you run a repair, preview the repair to make sure the operations that would be performed are correct. To preview the repair operations, use the **–check** option. All repair operations are displayed to stderr, but not performed. In an active system, however, the operations displayed by the **–check** option might not be the same as the operations performed when you later run the repair.

The server on which you run the **cdr repair** command must have a copy of the ATS or RIS file and be able to connect to the source and target database servers involved in the failed transaction. In a hierarchical routing environment where the source and target database servers are not directly connected you might need to run the **cdr repair** command from an intermediate server. If necessary, copy the ATS or RIS file to the intermediate server.

ATS and RIS files do not include code set information, therefore, the code sets associated with the locales specified by the DB_LOCALE and CLIENT_LOCALE environment variables must be the same.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example repairs inconsistencies between the **g_beijing** and **g_amsterdam** servers resulting from an aborted transaction:

```
% cdr repair ats ats.g_beijing.g_amsterdam.D_2.070827_12:58:55.1
Attempting connection to syscdr@amsterdam...
Using syscdr@amsterdam.
Source ID:10 / Name:g_amsterdam
Target ID:20 / Name:g_beijing
(1) [s = "1"]: Row will be updated on the source for replicate <655361>
(2) [s = "2"]: Row not found on the source for replicate <655361>
(2) [s = "2"]: Row not found on the target for replicate <655361>
(2) [s = "2"]: No operation will be performed.
(3) [s = "3"]: Row will be updated on the source for replicate <655361>
(4) [s = "4"]: Row will be updated on the source for replicate <655361>
(5) [s = "5"]: Row will be updated on the source for replicate <655361>
(6) [s8 = "1911"]: Row will be updated on the source for replicate <655362>
(7) [s8 = "1912"]: Row will be updated on the source for replicate <655362>
(8) [s8 = "1913"]: Row will be updated on the source for replicate <655362>
(9) [s8 = "1914"]: Row will be updated on the source for replicate <655362>
(10) [s8 = "1915"]: Row will be updated on the source for replicate <655362>
```

Related concepts:

"Preparing for Role Separation (UNIX)" on page 4-22

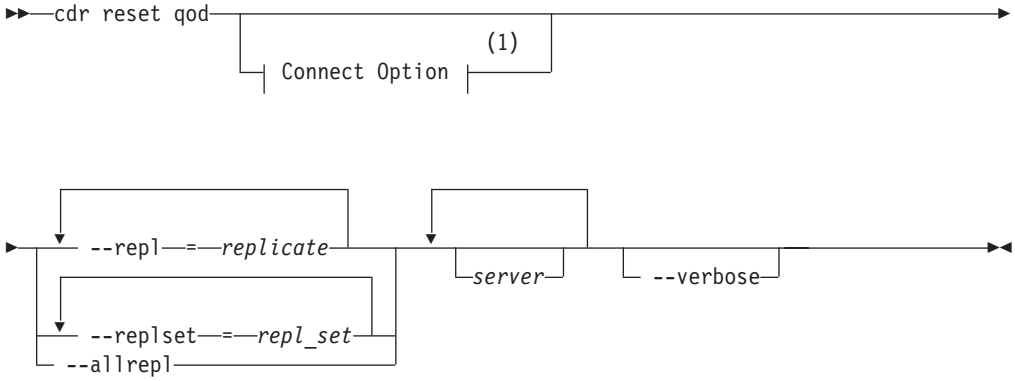
Related reference:

"cdr view" on page A-169

cdr reset qod

The **cdr reset qod** command resets the failed transaction count for replication servers that are using a Connection Manager SLA with the FAILURE or LATENCY policies.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
replicate	The name of the replicate.	The replicate must exist.	"Long Identifiers" on page A-3
repl_set	The name of the replicate set.	The replicate set must exist.	"Long Identifiers" on page A-3

Element	Purpose	Restrictions	Syntax
server	The name of the server.	Must be the name of an existing database server group in SQLHOSTS. Cannot be a leaf server.	"Long Identifiers" on page A-3

The following table describes the options to the **cdr reset qod** command.

Long Form	Short Form	Meaning
--allrepl	**-A**	Resets the failed transaction count on all replicates.
--repl=	**-r**	Specifies the replicate for which to reset the failed transaction count.
--replset=	**-s**	Specifies the replicate set for which to reset the failed transaction count.
--verbose	**-v**	Displays details of the operations the command is performing

Usage

Use the **cdr reset qod** command to reset the failed transaction count to zero for the specified replicates or replicate sets on the specified servers. Run the **cdr reset qod** command before you repair inconsistent data to count any failures that occur after the repair.

You must run the **cdr reset qod** command from a non-leaf server. If you do not specify any servers to reset, the current server to which you are connected is reset. If you specify one or more servers to reset, you must explicitly include the server to which you are connected if you want to reset it.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 44, 217.

For information on error codes, see "Return Codes for the cdr Utility" on page A-8.

Example

The following example resets the failed transaction count for all replicates on the server **gserv1** and **gserv2**:

```
cdr reset qod --allrepl gserv1 gserv2
```

The following example resets the failed transaction count for the **repl1** replicate on the server **gserv1**:

```
cdr reset qod --repl=repl1 gserv1
```

Related tasks:

"Routing client connections in a grid" on page 7-16

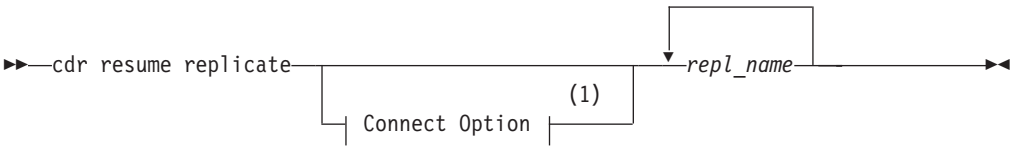
Related reference:

"cdr start qod" on page A-130

cdr resume replicate

The **cdr resume replicate** command resumes delivery of replication data.

Syntax



Notes:

1 See "Connect Option" on page A-3.

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-3

Usage

The **cdr resume replicate** command causes all participants in the replicate *repl_name* to enter the active state.

If a replicate belongs to an exclusive replicate set, 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.

When you run the **cdr resume replicate** command, an event alarm with a class ID of 57 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server (the one specified by the **INFORMIXSERVER** environment variable) and resumes the replicate **smile**:

```
cdr res repl smile
```

Related tasks:

"Exclusive Replicate Sets" on page 6-18

Related reference:

"cdr change replicate" on page A-32

"cdr define replicate" on page A-61

"cdr delete replicate" on page A-79

"cdr list replicate" on page A-97

"cdr modify replicate" on page A-108

"cdr start replicate" on page A-131

"cdr stop replicate" on page A-152

"cdr suspend replicate" on page A-155

"Enterprise Replication Event Alarms" on page 9-21

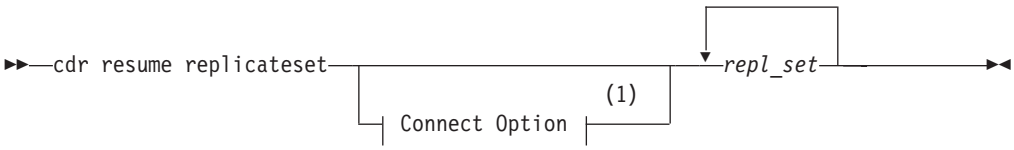
"cdr list server" on page A-103

"cdr resume replicateset"

cdr resume replicateset

The **cdr resume replicateset** command resumes delivery of replication data for all the replicates in a replicate set.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to resume.	None.	"Long Identifiers" on page A-3

Usage

The **cdr resume replicateset** command causes all replicates contained in the replicate set *repl_set* to enter the active state for all participants.

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.

When you run the **cdr resume replicateset** command, an event alarm with a class ID of 58 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server (the one specified by the **INFORMIXSERVER** environment variable) and resumes the replicate set **accounts_set**:

```
cdr res replset accounts_set
```

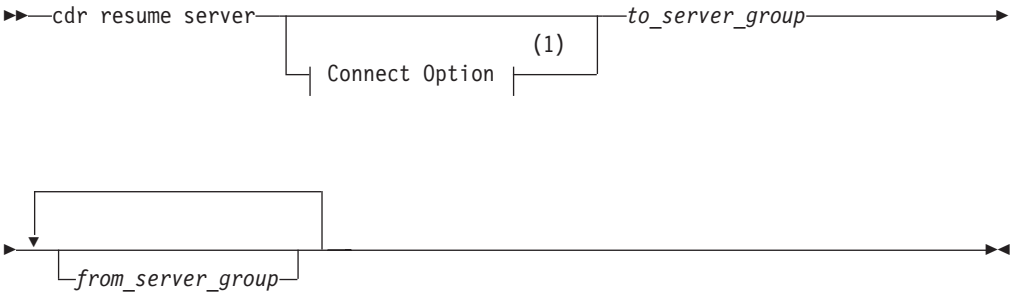
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr delete replicateset" on page A-80
"cdr list replicateset" on page A-101
"cdr modify replicateset" on page A-111
"cdr define replicate" on page A-61
"cdr start replicateset" on page A-134
"cdr stop replicateset" on page A-154
"cdr suspend replicateset" on page A-157
"Enterprise Replication Event Alarms" on page 9-21
"cdr resume replicate" on page A-126
"cdr list server" on page A-103

cdr resume server

The **cdr resume server** command resumes delivery of replication data to a suspended database server.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions
to_server_group	Name of the 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.

When you run the **cdr resume server** command, an event alarm with a class ID of 52 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server (the one specified by the **INFORMIXSERVER** environment variable) and resumes replication of data to the server **g_iowa** from the servers **g_ohio** and **g_utah**:

```
cdr res serv g_iowa g_ohio g_utah
```

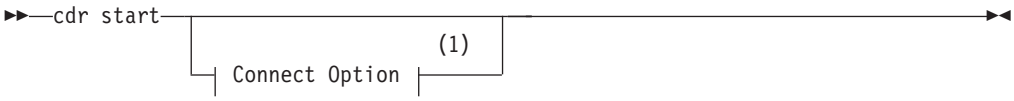
Related reference:

"cdr connect server" on page A-57
"cdr define server" on page A-71
"cdr delete server" on page A-81
"cdr disconnect server" on page A-88
"cdr list server" on page A-103
"cdr modify server" on page A-112
"cdr suspend server" on page A-158
"Enterprise Replication Event Alarms" on page 9-21

cdr start

The **cdr start** command starts Enterprise Replication processing.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Usage

Use **cdr start** to restart Enterprise Replication after you stop it with the **cdr stop** command or replication stops for another reason, such as memory allocation problems. When you issue **cdr start**, Enterprise Replication activates all connections to other connected replication servers. Replication servers, replicates, and replicate sets that were suspended before the **cdr stop** command was issued remain suspended; no data is sent for the suspended servers, replicates, or sets.

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 replication is stopped. When replication resumes, all appropriate database transactions that occurred while replication was stopped are replicated. If replication is stopped for a prolonged period of time, the replay position in the logical log might be overwritten. If the replay position is not available, the **cdr start** command fails with return code 214 and event alarm 75 is raised. In this situation, you must empty the send queues and reset the replay position by running the **cdr cleanstart** command, and then synchronize the data.

When you run the **cdr start** command, event alarm 49 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Important: Whenever replication is stopped, data can become inconsistent. Therefore, issue **cdr start** and **cdr stop** with extreme caution.

Examples

The following example restarts Enterprise Replication processing on database server **utah**:

```
cdr start -c utah
```

Related tasks:

"Restarting Replication on a Server" on page 8-4

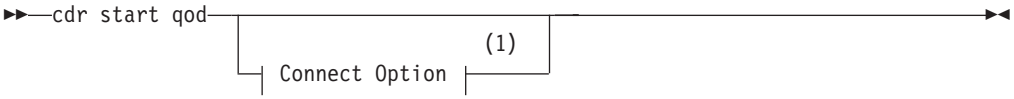
Related reference:

"cdr cleanstart" on page A-57
"cdr list server" on page A-103
"cdr stop" on page A-150
"Enterprise Replication Event Alarms" on page 9-21

cdr start qod

The **cdr start qod** command starts the monitoring of data quality.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Usage

Use the **cdr start qod** command to start monitoring the quality of replicated data for the replications servers in a grid so that a Connection Manager can use this information to decide how to route client connections to replication servers based

on the SLA. A master server for data quality must have already been defined by running the **cdr define qod** command. The **cdr start qod** command must be run from the master server for data quality.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 5, 217.

For information on error codes, see "Return Codes for the cdr Utility" on page A-8.

Example

The following example starts monitoring data quality:

```
cdr start qod
```

Related reference:

"cdr define qod" on page A-60

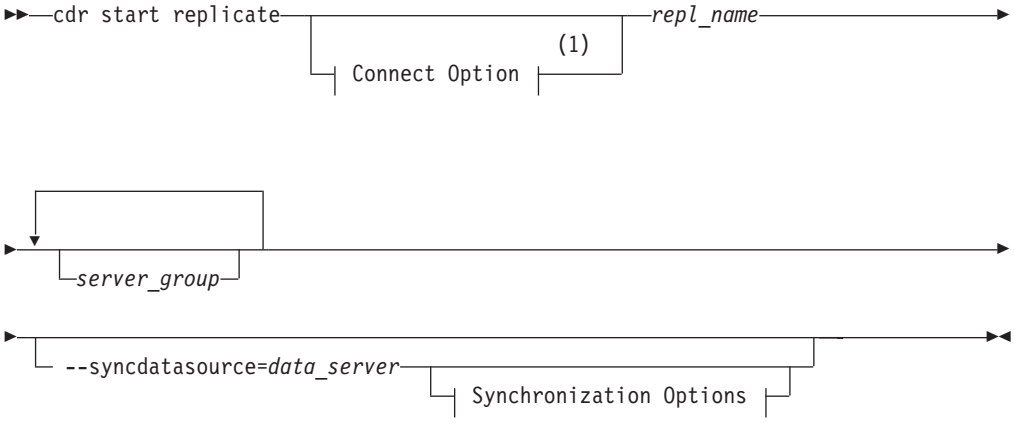
"cdr stop qod" on page A-152

"cdr reset qod" on page A-124

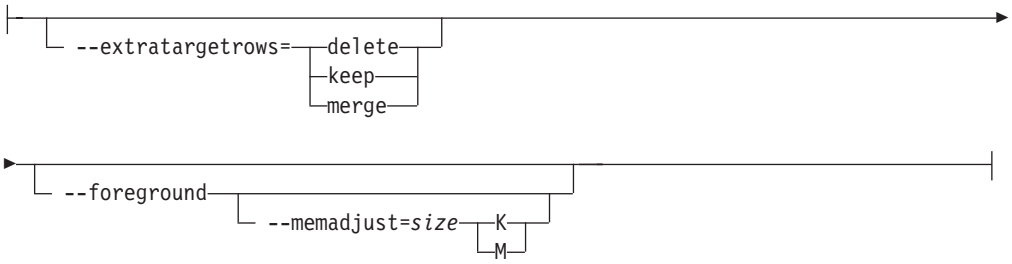
cdr start replicate

The **cdr start replicate** command starts the capture and transmittal of replication transactions.

Syntax



Synchronization Options:



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be in an Enterprise Replication domain.	
repl_name	Name of the replicate to start.	The replicate must exist.	"Long Identifiers" on page A-3
server_group	Name of database server groups on which to start the replicate.	The database server must be in an Enterprise Replication domain.	
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	

The following table describes the **cdr start replicate** options.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--syncdatasource=	**-S**	Specifies the name of the database server to use as the reference copy of the data. This server is started even if it is not listed as one of the servers to start.

Usage

The **cdr start replicate** command causes the replicate to enter the active state (capture-send) on the specified database servers and the source database server specified by the **--syncdatasource** option.

If you are running this command with the **--syncdatasource** option as a DBSA, you need to have certain permissions granted to you on several system tables in the **syscdr** database. For more information, see "Preparing for Role Separation (UNIX)" on page 4-22.

If you would like the synchronization operation to be run in the foreground, use the **--foreground** option.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

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, however, replication does not start until at least two participants are active. You cannot start replicates that have no participants.

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 was run, 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.

You can run the **cdr check queue --qname=cntrlq** command to wait for the **cdr start replicate** command to be applied at all Enterprise Replication servers before running the data synchronization task.

When you run the **cdr start replicate** command, an event alarm with a class ID of 59 is generated, if that event alarm is enabled.

Examples

The following command starts the replicate **accounts** on the server groups **g_svr1** and **g_svr2**:

```
cdr sta rep accounts g_svr1 g_svr2
```

The following example starts the replicate named **accounts** on the server **g_svr1** with **g_svr2** as the source server:

```
cdr start replicate accounts g_svr1 --syncdatasource=g_svr2\
--foreground --memadjust=50M
```

The second line indicates that the synchronization happens in the foreground and the size of the send queue is 50 MB.

Related concepts:

"Preparing for Role Separation (UNIX)" on page 4-22

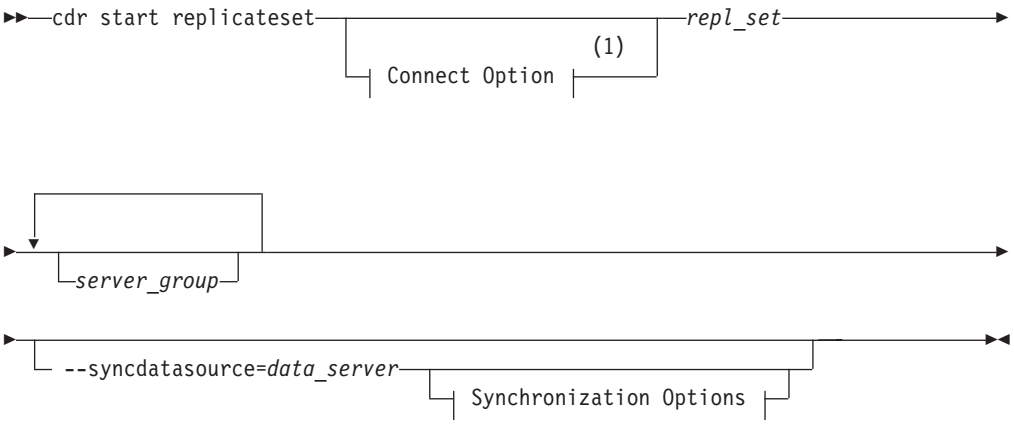
Related reference:

"cdr change replicate" on page A-32
"cdr define replicate" on page A-61
"cdr delete replicate" on page A-79
"cdr list replicate" on page A-97
"cdr modify replicate" on page A-108
"cdr resume replicate" on page A-126
"cdr stop replicate" on page A-152
"cdr suspend replicate" on page A-155
"Enterprise Replication Event Alarms" on page 9-21

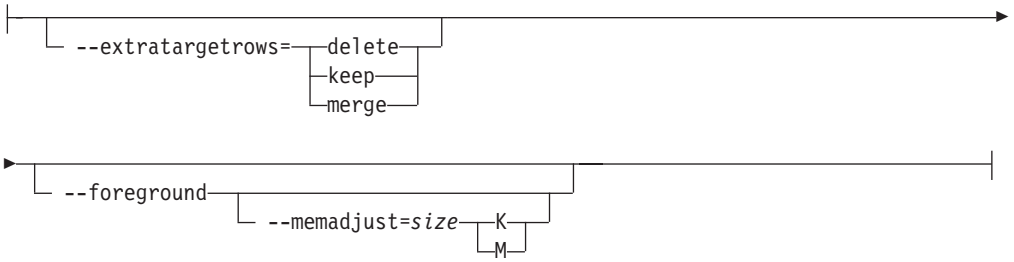
cdr start replicateset

The **cdr start replicateset** command starts the capture and transmittal of replication transactions for all the replicates in a replicate set.

Syntax



Synchronization Options:



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
data_server	The database server from which the data is copied to all other database servers listed.	The database server must be defined in Enterprise Replication.	
repl_set	Name of replicate set to start.	The replicate set must exist.	"Long Identifiers" on page A-3
server_group	Names of database server groups on which to start the replicate set.	The database server groups must be defined for Enterprise Replication.	
*size***K** or *size***M**	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	

The following table describes the **cdr start replicateset** options.

Long Form	Short Form	Meaning
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the data source server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers • **keep**: retain rows on the target servers • **merge**: retain rows on the target servers and replicate them to the data source server This option applies to the initial data synchronization operation only; it does not affect the behavior of the replicate.
--foreground	**-F**	Specifies that the synchronization operation is performed as a foreground process
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element
--syncdatasource=	**-S**	Specifies the name of the database server to use as the reference copy of the data. This server is started even if it is not listed as one of the servers to start.

Usage

The **cdr start replicateset** command causes the replicates defined in the specified replicate set to enter the active state (capture-send) on the specified database servers and the source database server specified by the **--syncdatasource** option.

If you are running this command with the **--syncdatasource** option as a DBSA, you need to have certain permissions granted to you on several system tables in the **syscdr** database. For more information, see "Preparing for Role Separation (UNIX)" on page 4-22.

If you would like the synchronization operation to be run as in the foreground, use the **--foreground** option.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

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 REPLICATION statement until after you successfully start the replicates in the replicate set.

If not all the replicates in a non-exclusive replicate set are inactive, the **cdr start replicateset** command displays a warning and only starts the replicates that are currently inactive.

When you run the **cdr start replicateset** command, an event alarm with a class ID of 60 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server specified by the **INFORMIXSERVER** environment variable 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
```

The following example starts the replicate set **accounts_set** on the server **g_hill** with **g_lake** as the source server:

```
cdr start replicateset accounts_set g_hill --syncdatasource=g_lake\
--foreground --memadjust=50M
```

The second line indicates that the synchronization happens in the foreground and the size of the send queue is 50 MB.

Related concepts:

"Preparing for Role Separation (UNIX)" on page 4-22

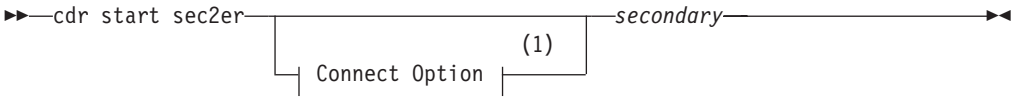
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr delete replicateset" on page A-80
"cdr list replicateset" on page A-101
"cdr modify replicateset" on page A-111
"cdr resume replicateset" on page A-127
"cdr define replicate" on page A-61
"cdr stop replicateset" on page A-154
"cdr suspend replicateset" on page A-157
"Enterprise Replication Event Alarms" on page 9-21

cdr start sec2er

The **cdr start sec2er** command converts a high availability cluster to replication servers.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
secondary	Name of the secondary server in the cluster.		"Long Identifiers" on page A-3

Usage

You must run the **cdr start sec2er** command from a primary server in a cluster with a high-availability data replication secondary or a remote stand-alone secondary server. The **cdr start sec2er** command converts the two cluster servers into replication servers.

The following conditions must be met on both the primary and secondary cluster servers before running the **cdr start sec2er** command:

- The `sqlhosts` files must be configured for Enterprise Replication:
  - Each server must belong to a different group.
  - The group identifier for each server must be different.
  - The `sqlhosts` files on each server must contain a server and a group entry for the other server.
- All databases and tables must be logged.
- No tables can be defined with label-based access control.

- Typed tables must have primary keys.
- User-defined types must support Enterprise Replication.
- The CDR_QDATA_SBSPACE configuration parameter must be set.
- Both server must be running Informix version 11.10 or later.
- If the servers are running Informix database software prior to 11.70, Enterprise Replication cannot be defined.
- Enterprise Replication must be active if it is already defined on either of the servers.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

The **cdr start sec2er** command performs the following tasks:

- The servers are defined as replication servers.
- Any tables on the primary server that do not have a primary key are altered to add ERKEY shadow columns.
- A replicate is created and started for each user table on the primary server.

If the **cdr start sec2er** command fails or is interrupted, you might see the following error message:

```
ERROR:  Command cannot be run on pre-11.70 instance if ER is already running.
```

If you receive this error, remove replication by running the **cdr delete server** command for both servers and then run the **cdr start sec2er** command again.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error codes is returned: 225.

For information on these error codes, see “Return Codes for the cdr Utility” on page A-8.

Example

The following example converts a cluster consisting of a primary server named **priserv** and a secondary server named **secserv** into replication servers. The output of the **cdr start sec2er** command shows the commands that are run.

```
$cdr start sec2er secserv
--
--  Define ER for the first time
--
cdr define serv -c priserv -I priserv

--
--  Creating Replication Key
--
dbaccess - - <<EOF
database stores_demo;
alter table 'bill'.classes add ERKEY;
EOF

--
--  Define the replicates
--
```

```
--
--  Defining Replicates for Database stores_demo
--
cdr define repl --connect=priserv sec2er_1_1282611664_call_type --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.call_type" \
         "select * from 'bill'.call_type"
cdr start repl --connect=priserv sec2er_1_1282611664_call_type

cdr define repl --connect=priserv sec2er_2_1282611664_cust_calls --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.cust_calls" \
         "select * from 'bill'.cust_calls"
cdr start repl --connect=priserv sec2er_2_1282611664_cust_calls

. . .

cdr define repl --connect=priserv sec2er_5_1282611664_customer --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.customer" \
         "select * from 'bill'.customer"
cdr start repl --connect=priserv sec2er_5_1282611664_customer

cdr define repl --connect=priserv sec2er_6_1282611664_classes --master=priserv \
   --conflict=always --scope=row \
   "stores_demo@priserv:'bill'.classes" \
         "select * from 'bill'.classes"
cdr start repl --connect=priserv sec2er_6_1282611664_classes
--
--  Starting RSS to ER conversion
--
--
--  WARNING:
--
--  DDL statements will not be automatically propagated to the ER server
--  after converting the secondary server into an ER server.  If you
--  create or alter any objects, such as databases, tables, indexes, and
--  so on, you must manually propagate those changes to the ER node and
--  change any replication rules affecting those objects.
--
```

Related concepts:

"Considerations for Replicating Opaque Data Types" on page 2-17

"Preparing Logging Databases" on page 4-22

Related tasks:

"Configuring port and service names for replication servers" on page 4-2

Related reference:

"CDR_QDATA_SBSPACE Configuration Parameter" on page B-12

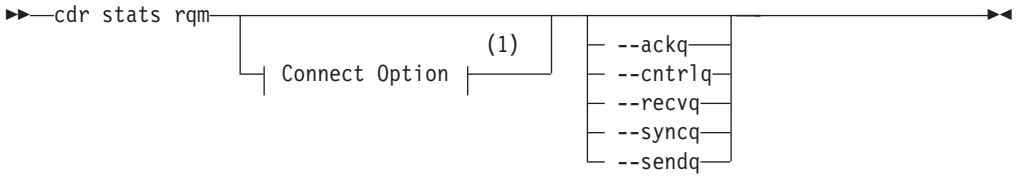
"cdr check sec2er" on page A-54

"Example of creating a new replication domain by cloning" on page 6-3

cdr stats rqm

The **cdr stats rqm** command displays information about the reliable queue manager (RQM) queues used for Enterprise Replication.

Syntax



Notes:

1 See "Connect Option" on page A-3.

The following table describes the **cdr stats rqm** options.

Long Form	Short Form	Meaning
--ackq	**-A**	Prints the statistics for the acknowledgment send queue.
--cntrlq	**-C**	Prints the statistics for the control send queue.
--recvq	**-R**	Prints the statistics for the receive queue.
--syncq	**-S**	Prints the statistics for the sync send queue.
--sendq	**-T**	Prints the statistics for the send queue.

Usage

The **cdr stats rqm** command displays the RQM (reliable queue manager) statistics for the queues used by Enterprise Replication. These queues are the ack send, control send, send, sync send, and the receive queue. If no queue is specified, the **cdr stats rqm** command displays statistics for all Enterprise Replication queues.

The **cdr stats rqm** command shows, among other things, how many transactions are currently queued in memory and spooled, the size of the data in the queue, how much real memory is being used, pending transaction buffers and data, the maximum memory used for data and headers (overhead), and totals for the number of transactions queued, the number of transactions, the number of deleted transactions, and the number of transaction lookups that have occurred.

If the Connect option is specified, Enterprise Replication connects to the specified remote server and retrieves the statistics for its Enterprise Replication queues.

Examples

The following example shows the output for **cdr stats rqm --ackq**:

```
RQM Statistics for Queue number: 1 name:  ack_send
 Flags:                     ACKSEND_Q, SENDQ_MASK
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 44 Bytes
 Max Real memory hdrs used: 320 Bytes
 Total data queued:         120 Bytes
 Total Txns queued:         0
 Total Txns                 3
 Total Txns spooled:        0
```

```
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        3
 Total Txns duplicated:     0
 Total Txn Lookups:         8
```

The following example shows the output for **cdr stats rqm --cntrlq**:

```
RQM Statistics for Queue number: 2 name:  control_send
 Transaction Spool Name:    control_send_stxn
 Flags:                     CTRL_SEND_Q, STABLE, USERTXN, PROGRESS_TABLE,
                            NEED_ACK, SENDQ_MASK
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 185 Bytes
 Max Real memory hdrs used: 320 Bytes
 Total data queued:         185 Bytes
 Total Txns queued:         0
 Total Txns                 1
 Total Txns spooled:        1
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        1
 Total Txns duplicated:     0
 Total Txn Lookups:         4
```

The following example shows the output for **cdr stats rqm --recvq**:

```
RQM Statistics for Queue number: 4 name:  trg_receive
 Transaction Spool Name:    trg_receive_stxn
 Flags:                     RECV_Q, SPOOLED, PROGRESS_TABLE
 Txns in queue:             0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 0 Bytes
 Max Real memory hdrs used: 0 Bytes
 Total data queued:         0 Bytes
 Total Txns queued:         0
 Total Txns                 0
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        0
 Total Txns duplicated:     0
 Total Txn Lookups:         0
```

The following example shows the output for **cdr stats rqm --syncq**:

```
RQM Statistics for Queue number: 3 name:  sync_send
 Flags:                     SYNC_Q, NEED_ACK, SENDQ_MASK
 Txns in queue:             0
 Txns in memory:            0
```

```
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 0 Bytes
 Max Real memory hdrs used: 0 Bytes
 Total data queued:         0 Bytes
 Total Txns queued:         0
 Total Txns                 0
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        0
 Total Txns duplicated:     0
 Total Txn Lookups:         1131
```

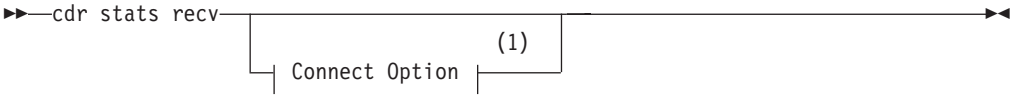
The following example shows the output for **cdr stats rqm --sendq**:

```
RQM Statistics for Queue number: 0 name:  trg_send
 Transaction Spool Name:    trg_send_stxn
 Flags:                     SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK,
                            SENDQ_MASK, SREP_TABLE
 Txns in queue:             12
 Txns in memory:            12
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           24960
 Size of Data in queue:     24960 Bytes
 Real memory in use:        24960 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 24960 Bytes
 Max Real memory hdrs used: 22080 Bytes
 Total data queued:         27560 Bytes
 Total Txns queued:         0
 Total Txns                 14
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        2
 Total Txns duplicated:     0
 Total Txn Lookups:         28
```

cdr stats recv

The **cdr stats recv** command displays receiver parallelism statistics and latency statistics by source node.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Usage

The **cdr stats recv** command displays the parallelism statistics for the receiver, including transaction count, number of pending and active transactions, the maximum that have been pending and active, the average number of pending and active transactions, and the commit rate. Totals and averages are calculated for pending and active transactions for the servers listed.

The Statistics by Source report shows the breakdown of transactions (number of inserts, updates, and deletes) and the latest source commit time and target apply time per server. The replication latency is the difference between the time when the transaction was committed on the source server and the time when the same transaction is applied on the target.

If the Connect option is specified, Enterprise Replication connects to the specified remote server and retrieved the statistics from it.

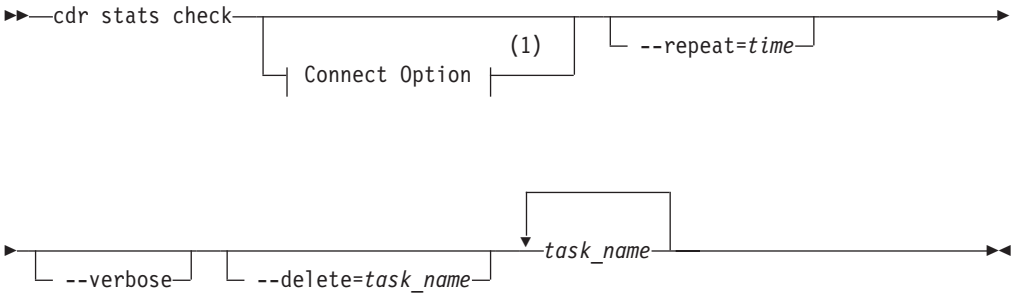
Examples

The following output is an example of the **cdr stats recv** command:

```
cdr stats recv

Receive Parallelism Statistics
Server Tot.Txn. Pending Active MaxPnd MaxAct  AvgPnd  AvgAct CommitRt
   144     11         0      0      3      2    1.27    1.36    0.01

Tot Pending:0       Tot Active:0    Avg Pending:0.00    Avg Active:0.00

Avg Commit Rate:0.01


Statistics by Source

Server  Repl     Txn   Ins   Del   Upd   Last Target Apply    Last Source Commit
144     9371650   11     0     0   220   2005/03/30 09:36:25  2005/03/30 09:36:25
```

cdr stats check

The **cdr stats check** command displays the progress of a consistency check that specified a progress report task name.



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task to display.	Must be an existing named task.	"Long Identifiers" on page A-3
time	The number of seconds between progress reports.	Must be a positive integer.	

The following table describes the options to the **cdr stats check** command.

Long Form	Short Form	Meaning
--delete=	**-d**	Specifies to delete the specified named task information from the **replcheck_stat** and **replcheck_stat_node** tables.
--repeat=	**-r**	Specifies to repeat the progress report every specified interval of seconds.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr stats check** command to display the progress of a consistency check operation while the **cdr check replicate** or **cdr check replicateset** command is running. You must have specified a task name in the **cdr check replicate** or **cdr check replicateset** command. You must be connected to the same server on which the **cdr check replicate** or **cdr check replicateset** command was run when you run the **cdr stats check** command.

The **cdr stats check** command displays a snapshot of the consistency report and an estimate of the time remaining to complete the consistency check. If you use the **--repeat** option, the consistency report is displayed every specified time interval.

You can view the progress of previously run consistency checks that have named tasks, if those progress report tasks have not been overwritten or deleted.

If you want to see the detailed progress report, include the **--verbose** option. The format of the verbose progress report is the same as the verbose consistency report generated by the **cdr check replicate** and **cdr check replicateset** commands.

If you want to delete a named task, use the **--delete** option.

Examples

The following example checks a replicate named **repl1**, creates a task named **tst**, and then displays a progress report every two seconds.

```
cdr check repl –r repl1 –m cdr1 –a --name=tst
cdr stats check –-repeat=2 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                    Started Jan 17 16:10:59
*********+----+----+----+----+----+----+----+----+ Remaining 0:00:08
```

```
------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
**********----+----+----+----+----+----+ Remaining 0:00:04

------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
**********----+----+----+----+ Remaining 0:00:02

------------------------------------------------
Job tst
repl1                    Started Jan 17 16:10:59
**********----+----+----+----+----+----+ Remaining 0:00:01

------------------------------------------------
Job tst
repl1                    Completed
                         Started Jan 17 16:10:59, Elapsed Time 0:00:07
```

The following example checks and repairs the replicate, creates a task named **tst**, and displays a verbose progress report every four seconds.

```
cdr check repl –r repl1 –m cdr1 –a --name=tst --repair
cdr stats check ––repeat=4 ––verbose tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                    Started Jan 17 16:34:42
********--+----+----+----+----+----+----+----+----+----+ Remaining 0:00:12

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                  9000         0         0         0         0         0
cdr2                  9000         0         0        99         0        99
cdr3                  9000         0         0         0         0         0

------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
**********----+----+----+----+ Remaining 0:00:02

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 43000         0         0         0         0         0
cdr2                 43000         0         0        99         0        99
cdr3                 43000         0         0         0         0         0

------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
**********----+----+----+----+----+----+----+----+----+----+ Remaining 0:00:01

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 39000         0         0         0         0        99
cdr2                 38901         0        99        99         0        99
cdr3                 39000         0         0         0         0         0

------------------------------------------------
Job tst
repl1                    Completed
                         Started Jan 17 16:34:42, Elapsed Time 0:00:11

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
```

```
cdr1                64099         0         0         0         0        99
cdr2                64000         0        99        99         0        99
cdr3                64099         0         0         0         0         0
```

The following example checks a replicate set named **set**, creates a task named **tst**, and displays a progress report every five seconds:

```
cdr check replset –s set –m cdr1 –a –n tst
cdr stats check –r 5 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl3                   Started Jan 17 16:41:19
*******----+----+----+----+----+----+----+----+----+ Remaining 0:00:16
repl2                   Pending
repl1                   Pending
Estimated time remaining for job tst  0:00:52

--------------------------------------------------
Job tst
repl3                   Started Jan 17 16:41:19
****************************************+----+----+ Remaining 0:00:01
repl2                   Pending
repl1                   Pending
Estimated time remaining for job tst  0:00:19

--------------------------------------------------
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Started Jan 17 16:41:27
********************+----+----+----+----+----+----+ Remaining 0:00:06
repl1                   Pending
Estimated time remaining for job tst  0:00:13

--------------------------------------------------
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Completed
                        Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                   Started Jan 17 16:41:35
----+----+----+----+----+----+----+----+----+----+----+ Remaining 0:01:08
Estimated time remaining for job tst  0:01:08

--------------------------------------------------
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Completed
                        Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                   Started Jan 17 16:41:35
********************************+----+----+----+----+ Remaining 0:00:02
Estimated time remaining for job tst  0:00:02

--------------------------------------------------
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Completed
                        Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                   Completed
                        Started Jan 17 16:41:35, Elapsed Time 0:00:11
Run time for job tst  0:00:27
```

Related tasks:

"Checking Consistency and Repairing Inconsistent Rows" on page 8-17

Related reference:

"The replcheck_stat Table" on page F-1

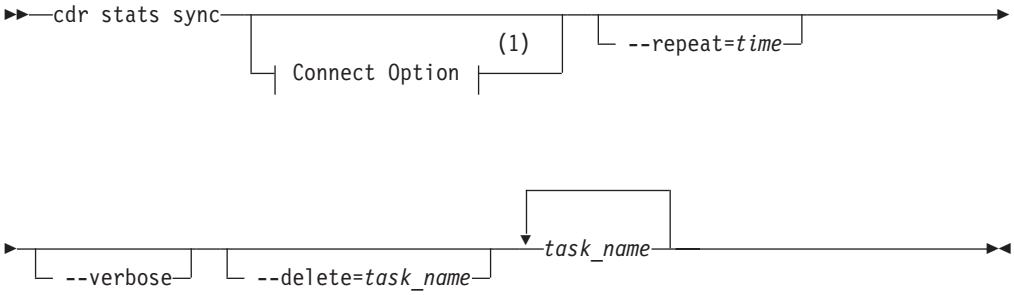
"The replcheck_stat_node Table" on page F-2

"cdr check replicate" on page A-37

"cdr check replicateset" on page A-47

cdr stats sync

The **cdr stats sync** command displays the progress of a synchronization operation that specified a progress report task name.



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
task_name	The name of the progress report task to display.	Must be an existing named task.	"Long Identifiers" on page A-3
time	The number of seconds between progress reports.	Must be a positive integer.	

The following table describes the options to the **cdr stats sync** command.

Long Form	Short Form	Meaning
--delete=	**-d**	Specifies to delete the specified named task information from the **replcheck_stat** and **replcheck_stat_node** tables.
--repeat=	**-r**	Specifies to repeat the progress report every specified interval of seconds.
--verbose	**-v**	Specifies that the consistency report shows specific values that are inconsistent instead of a summary of inconsistent rows.

Usage

Use the **cdr stats sync** command to display the progress of a synchronization operation (**cdr sync replicate** or **cdr sync replicateset**). You must be connected to the same server on which the **cdr sync replicate** or **cdr sync replicateset** command

was run when you run the **cdr stats sync** command. The **cdr stats sync** command displays a snapshot of the progress report and an estimate of the time remaining to complete the synchronization operation. If you use the **--repeat** option, the progress report is displayed every specified time interval.

You can view the progress of previously run synchronization operations that have named tasks, if those progress report tasks have not been overwritten or deleted.

If you want to see the detailed progress report, include the **--verbose** option. The format of the verbose progress report is the same as the verbose consistency report generated by the **cdr check replicate** and **cdr check replicateset** commands.

If you want to delete a named task, use the **--delete** option.

Examples

The following example synchronizes a replicate named **repl1**, creates a task named **tst**, and then displays a progress report every two seconds.

```
cdr sync repl –r repl1 –m cdr1 –a --name=tst
cdr stats sync ––repeat=2 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                   Started Jan 17 16:10:59
*********+----+----+----+----+----+----+----+----+ Remaining 0:00:08

--------------------------------------------------
Job tst
repl1                   Started Jan 17 16:10:59
*********************--+----+----+----+----+----+ Remaining 0:00:04

--------------------------------------------------
Job tst
repl1                   Started Jan 17 16:10:59
***********************************----+----+----+ Remaining 0:00:02

--------------------------------------------------
Job tst
repl1                   Started Jan 17 16:10:59
************************************************+ Remaining 0:00:01

--------------------------------------------------
Job tst
repl1                   Completed
                        Started Jan 17 16:10:59, Elapsed Time 0:00:07
```

The following example synchronizes the replicate, creates a task named **tst**, and displays a verbose progress report every four seconds.

```
cdr sync repl –r repl1 –m cdr1 –a --name=tst
cdr stats sync ––repeat=4 ––verbose tst
```

The progress report from the previous command might look like this:

```
Job tst
repl1                   Started Jan 17 16:34:42
*******--+----+----+----+----+----+----+----+----+ Remaining 0:00:12

Node                Total     Extra   Missing  Mismatch     Child Processed
--------------- --------- --------- --------- --------- --------- ---------
cdr1                 9000         0         0         0         0         0
cdr2                 9000         0         0        99         0        99
cdr3                 9000         0         0         0         0         0
```

```
------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
********************************+----+----+----+ Remaining 0:00:02

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 43000         0         0         0         0         0
cdr2                 43000         0         0        99         0        99
cdr3                 43000         0         0         0         0         0

------------------------------------------------
Job tst
repl1                    Started Jan 17 16:34:42
************************************************+ Remaining 0:00:01

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 39000         0         0         0         0        99
cdr2                 38901         0        99        99         0        99
cdr3                 39000         0         0         0         0         0

------------------------------------------------
Job tst
repl1                    Completed
                         Started Jan 17 16:34:42, Elapsed Time 0:00:11

Node                 Total     Extra   Missing  Mismatch     Child Processed
---------------- --------- --------- --------- --------- --------- ---------
cdr1                 64099         0         0         0         0        99
cdr2                 64000         0        99        99         0        99
cdr3                 64099         0         0         0         0         0
```

The following example synchronizes a replicate set named **set**, creates a task named **tst**, and displays a progress report every five seconds:

```
cdr sync replset –s set –m cdr1 –a –n tst
cdr stats sync –r 5 tst
```

The progress report from the previous command might look like this:

```
Job tst
repl3                    Started Jan 17 16:41:19
******----+----+----+----+----+----+----+----+----+ Remaining 0:00:16
repl2                    Pending
repl1                    Pending
Estimated time remaining for job tst  0:00:52

------------------------------------------------
Job tst
repl3                    Started Jan 17 16:41:19
****************************************+----+----+ Remaining 0:00:01
repl2                    Pending
repl1                    Pending
Estimated time remaining for job tst  0:00:19

------------------------------------------------
Job tst
repl3                    Completed
                         Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                    Started Jan 17 16:41:27
******************+----+----+----+----+----+----+ Remaining 0:00:06
repl1                    Pending
Estimated time remaining for job tst  0:00:13

------------------------------------------------
```

```
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Completed
                        Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                   Started Jan 17 16:41:35
----+----+----+----+----+----+----+----+----+----+ Remaining 0:01:08
Estimated time remaining for job tst  0:01:08

------------------------------------------------
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Completed
                        Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                   Started Jan 17 16:41:35
**********************************+----+----+----+ Remaining 0:00:02
Estimated time remaining for job tst  0:00:02

------------------------------------------------
Job tst
repl3                   Completed
                        Started Jan 17 16:41:19, Elapsed Time 0:00:08
repl2                   Completed
                        Started Jan 17 16:41:27, Elapsed Time 0:00:08
repl1                   Completed
                        Started Jan 17 16:41:35, Elapsed Time 0:00:11
Run time for job tst  0:00:27
```

Related tasks:

"Performing Direct Synchronization" on page 8-15

Related reference:

"The replcheck_stat Table" on page F-1

"The replcheck_stat_node Table" on page F-2

"cdr sync replicate" on page A-161

"cdr sync replicateset" on page A-164

cdr stop

The **cdr stop** command stops replication on the server to which you are connected without shutting down the database server.

Syntax

Notes:

1 See "Connect Option" on page A-3.

Usage

Generally, to stop replication on a server, you should shut down the database server. Under rare conditions, you might want to temporarily stop the Enterprise Replication processing without shutting down the database server.

The **cdr stop** command shuts down replication 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.

Stopping replication has the following effects:

- There is no connection between the stopped server and active replication servers.
- Transactions on the stopped server are not captured for replication. However, after restarting replication, transaction capture restarts at the replay position. If replication is stopped for long enough that the replay position is overwritten, you must restart replication with the **cdr cleanstart** command. If the CDR_LOG_LAG_ACTION configuration parameter is set to **logstage**, logs are staged to protect the replay position.
- Transactions on active replication servers are queued for the stopped server, but there is the possibility of filling up the send queues.
- Control messages on active replication servers are queued for the stopped server.
- The only Enterprise Replication commands you can run on the stopped server are **cdr start**, **cdr cleanstart**, and **cdr delete server** with the **--force** option.

To ensure consistency, prevent database update activity while Enterprise Replication is stopped. Replication threads remain stopped until you issue a **cdr start** command. Shutting down and restarting the stopped database server does not restart replication.

If you plan to stop replication for a long period of time and your replicates use time stamp or delete wins conflict resolution rules, consider using the **cdr disable server** command instead of the **cdr stop** command.

When you run the **cdr stop** command, event alarms with class IDs of 50 and 71 are generated, if those event alarms are enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Return Codes

0 The command was successful.

5 Enterprise Replication cannot connect to the specified server.

48 There is not enough memory to perform the operation.

62 Enterprise Replication is not active.

93 Enterprise Replication is in the process of starting.

94 Enterprise Replication is already in the process of stopping.

Examples

The following example stops Enterprise Replication processing on database server **paris**. Processing does not resume until a **cdr start** command restarts it:

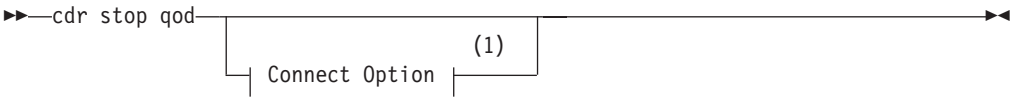
```
cdr stop -c paris
```

Related concepts:
"Resynchronizing Data among Replication Servers" on page 8-14
Related tasks:
"Temporarily stopping replication on a server" on page 8-3
Related reference:
"cdr start" on page A-129
"Enterprise Replication Event Alarms" on page 9-21

cdr stop qod

The **cdr stop qod** command stops the monitoring of data quality.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Usage

Use the **cdr stop qod** command to stop monitoring the quality of replicated data.

This command must be run as user **informix** (UNIX) or a member of the **Informix-Admin** group (Windows).

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, the following error code is returned: 217.

For information on this error code, see "Return Codes for the cdr Utility" on page A-8.

Example

The following example stops monitoring data quality:

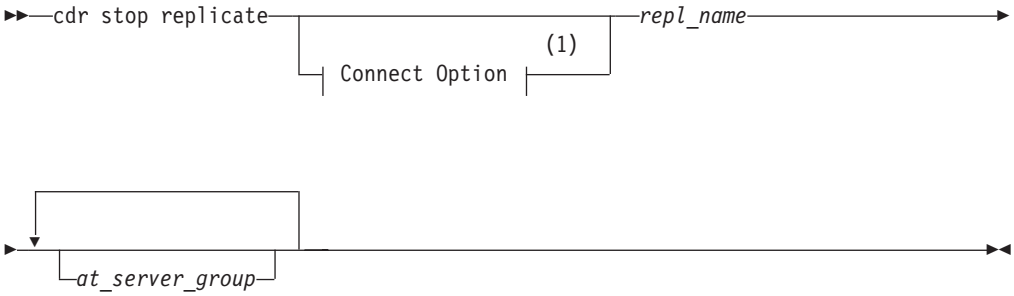
```
cdr stop qod
```

Related reference:
"cdr define qod" on page A-60
"cdr start qod" on page A-130

cdr stop replicate

The **cdr stop replicate** command stops the capture, transmittal, and reception of transactions for replication.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the new replicate.	The replicate must be active and not in an exclusive replicate set.	"Long Identifiers" on page A-3
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 replicated data is captured, sent or received) on the replication servers in the specified *at_server_group* list. In addition, this command deletes any data in the send queue for the stopped replicate. You cannot stop replicates that have no participants.

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.

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.

If you run this command while direct synchronization or consistency checking with repair is in progress, that repair process will stop. (Consistency checking continues; only the repair stops.) Direct synchronization and consistency checking repair cannot be resumed; you must rerun **cdr sync replicate** or **cdr check replicate** command with the **--repair** option.

When you run the **cdr stop replicate** command, an event alarm with a class ID of 61 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

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
```

Related concepts:

"Resynchronizing Data among Replication Servers" on page 8-14

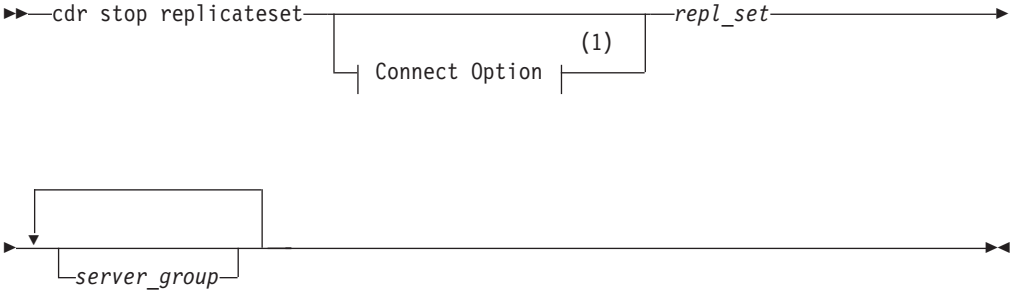
Related reference:

"cdr change replicate" on page A-32
"cdr define replicate" on page A-61
"cdr delete replicate" on page A-79
"cdr list replicate" on page A-97
"cdr modify replicate" on page A-108
"cdr resume replicate" on page A-126
"cdr start replicate" on page A-131
"cdr suspend replicate" on page A-155
"Enterprise Replication Event Alarms" on page 9-21

cdr stop replicateset

The **cdr stop replicateset** command stops capture and transmittal transactions for all the replicates in a replicate set.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to stop.	The replicate set must exist	"Long Identifiers" on page A-3
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.

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.

If you run this command while direct synchronization or consistency checking with repair is in progress, that repair process will stop. (Consistency checking continues; only the repair stops.) Direct synchronization and consistency checking repair cannot be resumed; you must rerun **cdr sync replicate** or **cdr check replicate** command.

When you run the **cdr stop replicateset** command, an event alarm with a class ID of 62 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

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
```

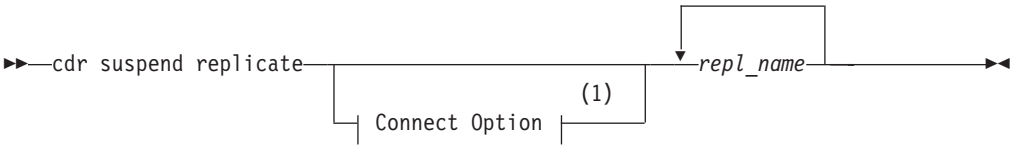
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr delete replicateset" on page A-80
"cdr list replicateset" on page A-101
"cdr modify replicateset" on page A-111
"cdr resume replicateset" on page A-127
"cdr start replicateset" on page A-134
"cdr define replicate" on page A-61
"cdr suspend replicateset" on page A-157
"Enterprise Replication Event Alarms" on page 9-21

cdr suspend replicate

The **cdr suspend replicate** command suspends delivery of replication data.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the replicate.	The replicate must be active.	"Long Identifiers" on page A-3

Usage

The **cdr suspend replicate** command causes the replicate *repl_name* to enter the suspend state (capture, no send) for all participants.

Attention: 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. 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.

If a replicate belongs to an exclusive replicate set, you cannot run **cdr suspend replicate** to suspend that individual replicate. You must use **cdr suspend replicateset** to suspend all replicates in the exclusive replicate set.

When you run the **cdr suspend replicate** command, an event alarm with a class ID of 55 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the database server **stan** and suspends the replicate **house**:

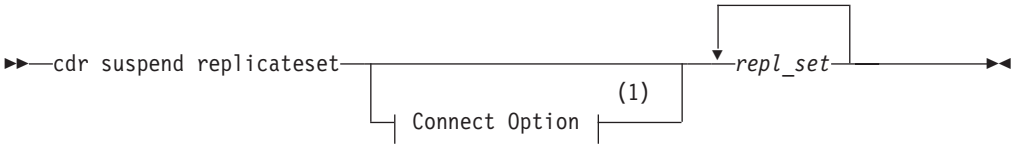
```
cdr sus repl --connect=stan house
```

Related reference:
"cdr change replicate" on page A-32
"cdr define replicate" on page A-61
"cdr delete replicate" on page A-79
"cdr list replicate" on page A-97
"cdr modify replicate" on page A-108
"cdr resume replicate" on page A-126
"cdr start replicate" on page A-131
"cdr stop replicate" on page A-152
"Enterprise Replication Event Alarms" on page 9-21
"cdr suspend replicateset"

cdr suspend replicateset

The **cdr suspend replicateset** command suspends delivery of replication data for all the replicates in a replicate set.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_set	Name of replicate set to suspend.	The replicate set must exist.	"Long Identifiers" on page A-3

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.

Attention: 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. 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.

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.

When you run the **cdr suspend replicateset** command, an event alarm with a class ID of 56 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default database server specified by **$INFORMIXSERVER** and suspends the replicate set **accounts_set**:

```
cdr sus replset account_set
```

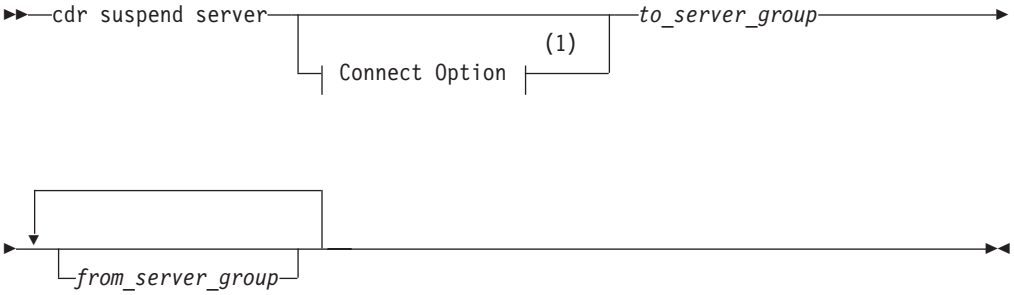
Related reference:

"cdr change replicateset" on page A-35
"cdr define replicateset" on page A-69
"cdr delete replicateset" on page A-80
"cdr list replicateset" on page A-101
"cdr modify replicateset" on page A-111
"cdr resume replicateset" on page A-127
"cdr start replicateset" on page A-134
"cdr stop replicateset" on page A-154
"cdr define replicate" on page A-61
"Enterprise Replication Event Alarms" on page 9-21
"cdr suspend replicate" on page A-155

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 domain.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions
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 replication domain to the *to_server_group*.

Suspending replication has the following effects:

- The connections between the suspended server and active replication servers remain active.
- Transactions on the suspended replication server are sent to the active replication servers.
- Transactions on active replication servers are queued for the suspended replication server.
- Control messages on active replication servers are sent to the suspended replication server.
- Control messages on the suspended replication server are sent to the active replication servers.

To restart replication on a suspended replication server, run the **cdr resume server** command. Shutting down and restarting the suspended database server does not resume replication.

When you run the **cdr suspend server** command, an event alarm with a class ID of 51 is generated, if that event alarm is enabled.

You can run this command from within an SQL statement by using the SQL administration API.

Examples

The following example connects to the default server (the one specified by the **INFORMIXSERVER** environment variable) 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
```

Related reference:

"cdr connect server" on page A-57

"cdr define server" on page A-71

"cdr delete server" on page A-81

"cdr disconnect server" on page A-88

"cdr list server" on page A-103

"cdr modify server" on page A-112

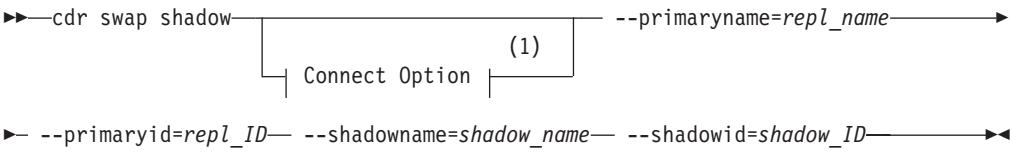
"cdr resume server" on page A-128

"Enterprise Replication Event Alarms" on page 9-21

cdr swap shadow

The **cdr swap shadow** command switches a replicate with its shadow replicate during manual remastering.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
repl_name	Name of the primary replicate.	The primary replicate participant attributes state, type (**P** or **R**), and table owner (**O** or **I**) must match the shadow replicate participant attributes.	"Long Identifiers" on page A-3
repl_ID	Internal Enterprise Replication identification code for the primary replicate.		
shadow_name	Name of the shadow replicate.	The shadow replicate state must match the primary replicate state. Shadow replicate participants must match the primary replicate participants.	"Long Identifiers" on page A-3
shadow_ID	Internal Enterprise Replication identification code for the shadow replicate.		

The following table describes the **cdr swap shadow** options.

Long Form	Short Form	Meaning
--primaryname=	**-p**	Specifies the name of the primary replicate.
--primaryid=	**-P**	Specifies the ID of the primary replicate.
--shadowname=	**-s**	Specifies the name of the shadow replicate.
--shadowid=	**-S**	Specifies the ID of the shadow replicate.

Usage

Use the **cdr swap shadow** command to switch a replicate with its shadow replicate as the last step in manually remastering a replicate that was created with the **--name=n** option. You create a shadow replicate using the **cdr define replicate** command with the **--mirrors** option.

Use the **onstat -g cat repls** command to obtain the *repl_ID* and *shadow_ID*. Alternatively, you can query the **syscdrrepl** view in the **sysmaster** database.

You can run this command from within an SQL statement by using the SQL administration API.

Related tasks:

"Remastering a Replicate" on page 8-27

"Enabling code set conversion between replicates" on page 6-14

Related reference:

"cdr alter" on page A-29

"cdr define replicate" on page A-61

"cdr list replicate" on page A-97

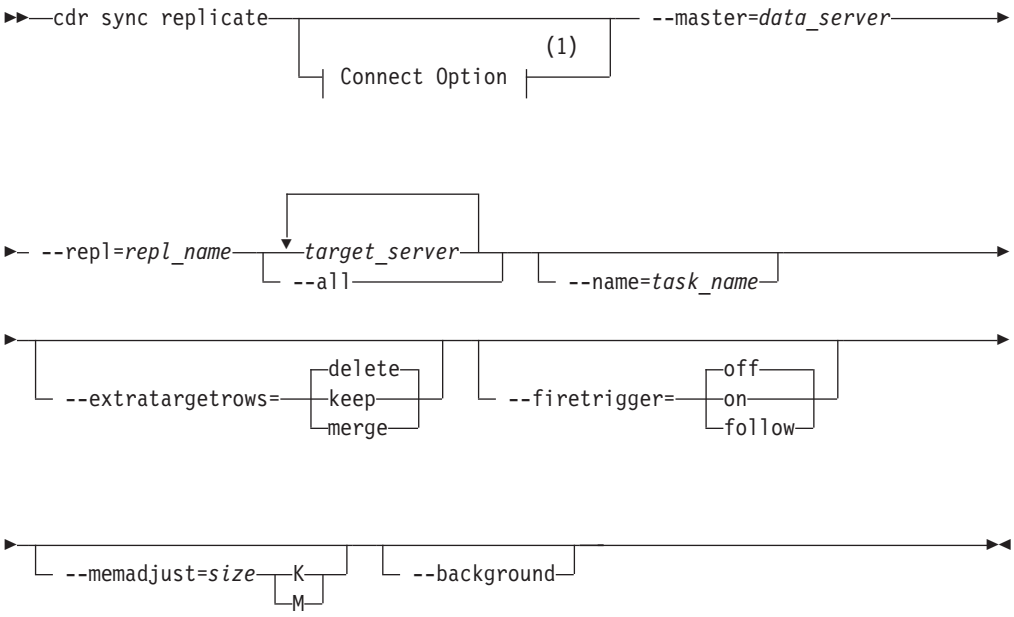
"Participant and participant modifier" on page A-4

"onstat -g rep" on page E-17

cdr sync replicate

The **cdr sync replicate** command synchronizes data among replication servers to repair inconsistent data within a replicate.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions	Syntax
data_server	Name of the database server to use as the reference copy of the data.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
repl_name	Name of the replicate to synchronize.		"Long Identifiers" on page A-3

Element	Purpose	Restrictions	Syntax
*size*K or *size*M	Size, in either kilobytes (**K**) or megabytes (**M**), of the send queue during synchronization.	Must be a positive integer and must not be greater than the amount of available memory.	
target_server	Name of a database server group on which to perform synchronization.	Must be the name of an existing database server group in SQLHOSTS.	"Long Identifiers" on page A-3
task_name	The name of the progress report task.	If you use an existing task name, the information for that task is overwritten. Maximum name length is 127 bytes.	"Long Identifiers" on page A-3

The following table describes the **cdr sync replicate** options.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are synchronized.
--background	**-B**	Specifies that the operation is performed in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name specified with the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers. • **keep**: retain rows on the target servers. • **merge**: retain rows on the target servers and replicate them to the data source server.
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while synchronizing the data: • **off**: (default) do not fire triggers at target servers during synchronization. • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option. • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option.
--master=	**-m**	Specifies the database server to use as the reference copy of the data.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.

Long Form	Short Form	Meaning
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--repl=	**-r**	Specifies the name of the replicate to synchronize.

Usage

Use the **cdr sync replicate** command to synchronize data between multiple database servers for a specific replicate. This command performs direct synchronization as a foreground process.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

If you want to monitor the progress of the synchronization operation, include the **--name** option and specify a name for the progress report task. Then run the **cdr stats sync** command and specify the progress report task name.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the **--name** option so that you can monitor the operation with the **cdr stats sync** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

The **cdr sync replicate** command performs the following tasks:

1. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
2. Performs a sequential scan of the replicated table on the source server.
3. Replicates the all rows in the table from the source server to the target server by copying the data directly into the send queue, bypassing the logical logs.
4. Deletes the shadow replicate.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 17, 18, 31, 37, 48, 53, 61, 75, 99, 101, 121, 172, 174, 178, 193, 194, 195, 200, 203, 204.

For information on these error codes, see “Return Codes for the cdr Utility” on page A-8

Examples

Example 1: Synchronize all servers

The following example illustrates synchronizing all replication servers for the replicate named **repl_1**:

```
cdr sync replicate --master=g_serv1 --repl=repl_1\
--all --extratargetrows=keep\
--firetrigger=on
```

The data on the server group **g_serv1** is used as the reference for correcting the data on the other servers. Line 2 indicates that all servers associated with the replicate are synchronized and that if the synchronization operation detects rows on the target servers that do not exist on the reference server (**g_serv1**), that those rows should remain on the other servers. Line 3 indicates that triggers should be fired on the target servers even if the replicate definition does not include the **--firetrigger** option.

Example 2: Synchronize three servers

The following example illustrates synchronizing three servers for the replicate named **repl_2**:

```
cdr sync replicate -m g_serv1 -r repl_2\
g_serv2 g_serv3
```

The reference server is **g_serv1** and the target servers are **g_serv2** and **g_serv3**. Because the **--extratargetrows** option is not specified, the default behavior occurs: rows, and any dependent rows that are based on referential integrity constraints, that are on the target servers but not on the reference server, are deleted.

Example 3: Synchronize in the background and set the send queue size

The following example illustrates synchronizing in the background and setting the size of the send queue to 50 MB:

```
cdr sync replicate --master=g_serv1 --repl=repl_1\
--memadjust=50M --background
```

Related concepts:

“Database Server Groups” on page 4-2

“Preparing for Role Separation (UNIX)” on page 4-22

Related tasks:

“Performing Direct Synchronization” on page 8-15

Related reference:

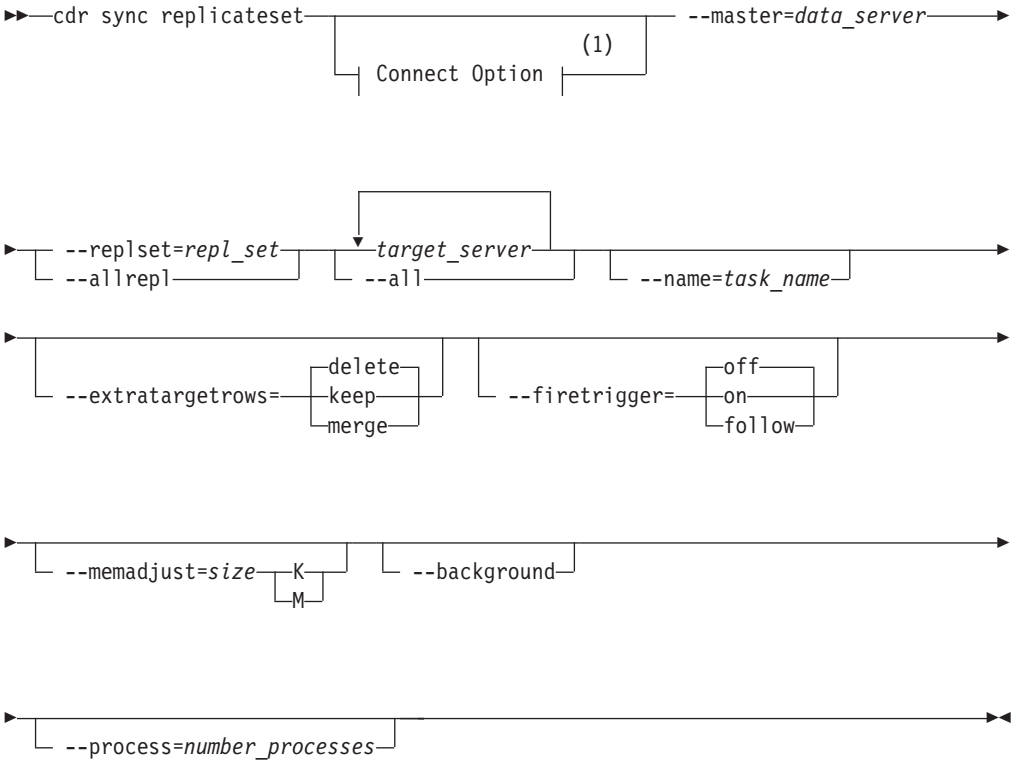
“cdr check replicate” on page A-37

“cdr stats sync” on page A-147

cdr sync replicateset

The **cdr sync replicateset** command synchronizes data among replication servers to repair inconsistent data within a replicate set.

Syntax



Notes:

1 See "Connect Option" on page A-3.

Long Form	Short Form	Meaning
--all	**-a**	Specifies that all servers defined for the replicate are synchronized.
--allrepl	**-A**	Specifies that all replicates are synchronized.
--background	**-B**	Specifies that the operation is performed in the background as an SQL administration API command. The command and its result are stored in the **command_history** table in the **sysadmin** database, under the name specified with the **--name=** option, or the time stamp for the command if **--name=** is not specified.
--extratargetrows=	**-e**	Specifies how to handle rows found on the target servers that are not present on the server from which the data is being copied (*data_server*): • **delete**: (default) remove rows and dependent rows, based on referential integrity constraints, from the target servers. • **keep**: retain rows on the target servers. • **merge**: retain rows on the target servers and replicate them to the data source server.

Long Form	Short Form	Meaning
--firetrigger=	**-T**	Specifies how to handle triggers at the target servers while synchronizing the data: • **off**: (default) do not fire triggers at target servers during synchronization. • **on**: always fire triggers at the target servers even if the replicate definition does not have the **--firetrigger** option. • **follow**: fire triggers at target servers only if the replicate definition has the **--firetrigger** option.
--master	**-m**	Specifies the database server to use as the reference copy of the data.
--memadjust=	**-J**	Increases the size of the send queue during synchronization to the number of kilobytes or megabytes specified by the *size* element.
--name=	**-n**	Specifies that the progress of this command can be monitored. Information about the operation is stored under the specified progress report task name on the server on which the command was run.
--process=	**-p**	Specifies to run the command in parallel, using the specified number of processes. At most, Enterprise Replication can use one process for each replicate in the replicate set. If you specify more processes than replicates, the extra processes are not used. Not all replicates can be processed in parallel. For example, if replicates have referential integrity rules, the replicates with the parent tables must be processed before the replicates with the child tables.
--replset	**-s**	Specifies the name of the replicate set to synchronize.

Usage

Use the **cdr sync replicateset** command to synchronize data between multiple database servers for a replicate set. This command performs direct synchronization as a foreground process.

If you run this command as a DBSA, you must have INSERT, UPDATE, and DELETE permission on the replicated tables on all the replication servers in the domain.

The size of the send queue is specified by the value of the CDR_QUEUEMEM configuration parameter. You can increase the amount of memory that the send queue can use during this synchronization operation by using the **--memadjust** option to specify the size of the send queue.

You can significantly improve the performance of synchronizing a replicate set by synchronizing the member replicates in parallel. You specify the number of parallel processes with the **--process** option. For best performance, specify the same number of processes as the number of replicates in the replicate set. However, replicates with referential integrity constraints cannot be processed in parallel.

If you want to monitor the progress of the synchronization operation, include the **--name** option and specify a name for the progress report task. Then run the **cdr stats sync** command and specify the progress report task name.

You can run a synchronization operation as a background operation as an SQL administration API command if you include the **--background** option. This option is useful if you want to schedule regular synchronization operations with the Scheduler. If you run a synchronization operation in the background, you should provide a name for the progress report task by using the **--name** option so that you can monitor the operation with the **cdr stats sync** command. You can also view the command and its results in the **command_history** table in the **sysadmin** database.

To synchronize all replicates at once, use the **--allrepl** option.

The **cdr sync replicateset** command performs the following tasks:

1. Determines the order in which to repair tables if they have referential relationships.
2. Creates a shadow replicate with the source server and target server as participants. The conflict resolution rule for the shadow replicate is always apply.
3. Performs a sequential scan of the replicated table on the source server.
4. Replicates the all rows in the table from the source server to the target server by copying the data directly into the send queue, bypassing the logical logs.
5. Deletes the shadow replicate.
6. Repeats steps 2 through 5 for each replicate in the replicate set.

You can run this command from within an SQL statement by using the SQL administration API.

Return codes

A return code of 0 indicates that the command was successful.

If the command is not successful, one of the following error codes is returned: 1, 5, 11, 17, 18, 31, 37, 48, 53, 61, 75, 99, 101, 121, 166, 172, 174, 193, 194, 195, 200, 203, 204, 213.

For information on these error codes, see "Return Codes for the cdr Utility" on page A-8

Examples

Example 1: Synchronize all servers

The following example illustrates synchronizing all replication servers for the replicate set **replset_1** using **g_serv1** as the reference server:

```
cdr sync replicateset --master=g_serv1 --replset=replset_1\
--all --extratargetrows=keep
```

Line 2 indicates that all servers associated with the replicate set are synchronized and that if the synchronization process detects rows on the target servers that do not exist on the reference server (**g_serv1**), that those rows should remain on the other servers.

Example 2: Synchronize three servers in parallel

The following example illustrates synchronizing three servers for the replicate set named **replset_2** and using two processes to synchronize each of the two replicates in the set in parallel:

```
cdr sync replicateset -m g_serv1 -s replset_2\
g_serv2 g_serv3 --process=2
```

The reference server is **g_serv1** and the target servers are **g_serv2** and **g_serv3**. Because the **--extratargetrows** option is not specified, the default behavior occurs: rows, and any dependent rows that are based on referential integrity constraints, that are on the target servers but not on the reference server, are deleted.

Example 3: Synchronize in the background and set the send queue size

The following example illustrates synchronizing in the background and setting the size of the send queue to 50 MB:

```
cdr sync replicateset --master=g_serv1 --replset=replset_1\
--memadjust=50M --background
```

Example 4: Synchronize all replicate sets on a replication server

The following command synchronizes all replicate sets on a replication server named **g_serv2**:

```
cdr sync replicateset --allrepl g_serv2
```

The replicate set name is not specified because the **--allrepl** option is used.

Related concepts:

"Database Server Groups" on page 4-2

"Preparing for Role Separation (UNIX)" on page 4-22

Related tasks:

"Performing Direct Synchronization" on page 8-15

Related reference:

"cdr check replicateset" on page A-47

"cdr stats sync" on page A-147

cdr -V

The **cdr -V** command displays the version of Informix that is currently running.

Syntax

Usage

Use the **cdr -V** command if you need to obtain the version of the database server, usually at the request of IBM Software Support.

Examples

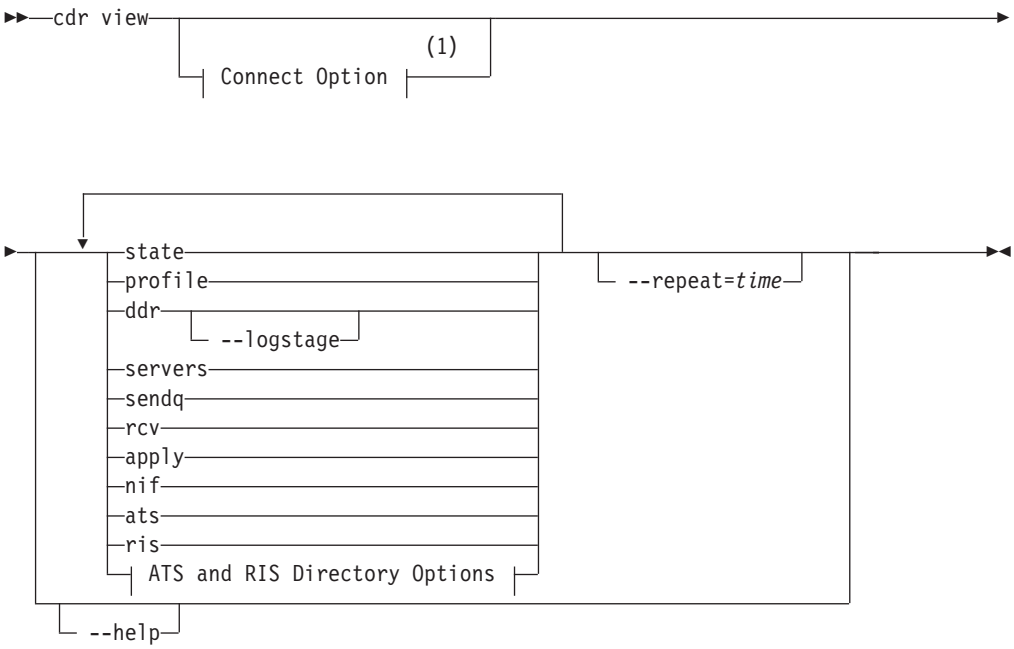
The following example shows an example output of the **cdr -V** command:

```
IBM Informix Version 11.70.UC1    Software Serial Number RDS#N000000
```

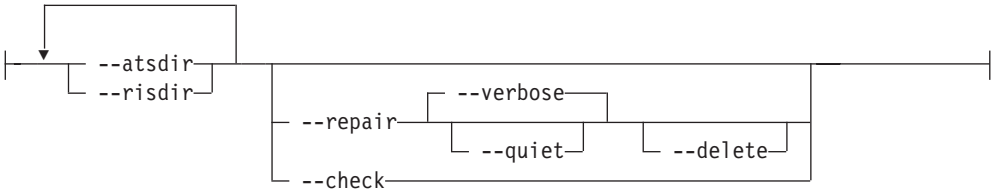
cdr view

The **cdr view** command displays information about every Enterprise Replication server in the domain.

Syntax



ATS and RIS Directory Options:



Notes:

1 See "Connect Option" on page A-3.

Element	Purpose	Restrictions
time	The number of seconds before the **cdr view** command is repeated.	Must be a positive integer.

The following table describes the **cdr view** subcommands.

Long Form	Meaning
apply	Display a summary of how data is being applied on each of the target servers, including the latency of each target server.
ats	Display a portion of each ATS file that is in text format.
atsdir	Display the names of the files in the ATS directory that are in text format and optionally run repair operations based on those files. If you are running this command as a DBSA, you must have read permission on the ATS files. Permissions on ATS files can be set with the **chown** operating system command.
ddr	Display the state, key log positions, and the proximity to transaction blocking for each server in the replication domain.
nif	Display information about the network connections between Enterprise Replication servers, including the number of transactions that are waiting to be transmitted to target servers.
profile	Display a summary of the state, data capture, data apply, errors, connectivity, queues, and the size of spooling files for every Enterprise Replication server.
rcv	Display information about the receive statistics for each target server, including the number of transaction failures and the rate at which transactions are applied.
ris	Display a portion of each RIS file that is in text format.
risdir	Display the names of the files in the RIS directory that are in text format and optionally run repair operations based on those files. If you are running this command as a DBSA, you must have read permission on the RIS files. Permissions on RIS files can be set with the **chown** operating system command.
sendq	Display information about the send queues for each Enterprise Replication server.
servers	Display information about the state, connection status to each peer server, and queue size for each Enterprise Replication server.
state	Display the Enterprise Replication state and the state of data capture, network connections, and data apply for each Enterprise Replication server.

The following table describes the **cdr view** options.

Long Form	Short Form	Meaning
--check	**-C**	Check the consistency between the database server and the ATS or RIS file. Display repair operations to stderr, but do not perform the repair operations.
--delete	**-d**	Delete ATS or RIS files after processing them with the repair operation.
--help	**-h**	Display the **cdr view** command usage.
–logstage	**-l**	Display log staging statistics.
--quiet	**-q**	Quiet mode. Repair operations are not displayed to stderr.

Long Form	Short Form	Meaning
--repair	**-R**	Synchronize data based on ATS or RIS files in text format.
--repeat=	**-r**	Repeat the **cdr view** command after the number of seconds specified by the *time* element.
--verbose	**-v**	Verbose mode (default). All repair operations are displayed to stderr.

Usage

Use the **cdr view** command to monitor the Enterprise Replication domain. Each subcommand results in different output information.

You can choose to display the output of multiple subcommands sequentially by including them in the same **cdr view** command. You can choose to automatically repeat the command by using the **--repeat** option to specify the seconds in between commands.

You can repair inconsistencies listed in ATS or RIS files on every server by using the **--repair** option. Use the **--delete** option to delete the ATS or RIS files after the repair is complete.

Tip: Using the **--repair** option is equivalent to running the **cdr repair** command. The **--check** option is equivalent to the **cdr repair --check** command.

The cdr view state Command Output

The following example of the output of the **cdr view state** command shows the state of Enterprise Replication and each of its main components for every server in the Enterprise Replication domain.

```
STATE
Source      ER              Capture          Network          Apply
            State           State            State            State
------------------------------------------------------------------------------
cdr1        Active          Running          Running          Running
cdr2        Active          Running          Running          Running
cdr3        Active          Running          Running          Running
cdr4        Active          Running          Running          Running
```

In this example, Enterprise Replication is active and running normally on all servers.

Possible values in the ER State column include:

Abort Enterprise Replication is aborting on this server.

Active Enterprise Replication is running normally.

Down Enterprise Replication is stopped.

Dropped
The attempt to drop the **syscdr** database failed.

Init Failed
The initial start-up of Enterprise Replication on this server failed, most likely because of a problem on the specified global catalog synchronization server.

Initializing
Enterprise Replication is being defined.

Initial Startup
Enterprise Replication is starting for the first time on this server.

Shutting Down
Enterprise Replication is stopping on this server.

Startup Blocked
Enterprise Replication cannot start because the server was started with the **oninit -D** command.

Synchronizing Catalogs
The server is receiving a copy of the **syscdr** database.

Uninitialized
The server does not have Enterprise Replication defined on it.

Possible values in the Capture State, Network State, and Apply State columns include:

Running
The Enterprise Replication component is running normally.

Down The Enterprise Replication component is not running.

Uninitialized
The server is not a source server for replication.

The cdr view profile Command Output

The following example of the output of the **cdr view profile** command shows a summary of the other **cdr view** commands and information about the sbspaces designated for spooled transaction data.

```
ER PROFILE for Node cdr2                ER State Active

DDR - Running                           SPOOL DISK USAGE
  Current        4:16879616               Total                  100000
  Snoopy         4:16877344               Metadata Free            5025
  Replay         4:24                     Userdata Free           93193
  Pages from Log Lag State   43879
                                        RECVQ
SENDQ                                     Txn In Queue                0
  Txn In Queue               0            Txn In Pending List         0
  Txn Spooled                0
  Acks Pending               0          APPLY - Running
                                          Txn Processed            1838
NETWORK - Running                         Commit Rate             76.58
  Currently connected to 3 out of 3       Avg. Active Apply        1.16
  Msg Sent                1841            Fail Rate                0.00
  Msg Received            5710            Total Failures              0
  Throughput           1436.94            Avg Latency              0.00
  Pending Messages           0            Max Latency                 0
                                          ATS File Count              0
                                          RIS File Count              0
```

In this example, only the output for a single server, **cdr2**, is shown. The actual output of the **cdr view profile** command includes a similar profile for every server.

The DDR section is a summary of the **cdr view ddr** command.

The SPOOL DISK USAGE section shows the total amount of memory, in bytes, in the sbspaces that Enterprise Replication uses to store spooled transaction row data, and the amount of available metadata and user data space.

The SENDQ section is a summary of the **cdr view sendq** command.

The RECVQ section is a summary of the **cdr view rcv** command.

The NETWORK section is a summary of the **cdr view nif** command.

The APPLY section is a summary of the **cdr view apply** command.

The cdr view ddr Command Output

The following example of the output of the **cdr view ddr** command shows the status of log capture.

```
Server   Snoopy       Replay       Current      total        log pages to   LogLag   Cur LogLag
         log page     log page     log page     log pages    LogLag State   State    Action
-------------------------------------------------------------------------------------------
g_bombay 16:133       16:0         16:134       30000        17866          Off       dlog
g_delhi  30:490       30:0         30:491       5000         3508           Off       logstage
```

The following example of the output of the **cdr view ddr -l** command shows the status of log capture.

```
Server   Disk Space   Max allowed  Max disk     Cur Staged
         Usage(%)     Space(KB)    ever used(KB) log file cnt
-----------------------------------------------------------------------------------------
g_bombay 0.00         0            0.00         0
g_delhi  0.00         1048576      0.00         0
```

The columns in the output of the **cdr view ddr** command provide the following information:

Server The name of the Enterprise Replication server.

Snoopy log page
The current log ID and position at which transactions are being captured for replication.

Replay log page
The current log ID and position at which transactions have been applied. This is the position from which the log would need to be replayed to recover Enterprise Replication if Enterprise Replication or the database server shut down.

Current log page
The log page on which replicated transactions are being captured.

total log pages
The total number of log pages on the server.

log pages to LogLag State
The number of log pages that would have to be used before transaction blocking occurs.

LogLag State
The state of DDR log lag: on or off.

Cur LogLag Action
The action being taken to catch up logs.

For more information on interpreting this output, see “onstat -g ddr” on page E-6.

The cdr view servers Command Output

The following example of the output of the **cdr view servers** command shows the state of the Enterprise Replication servers and their connections to each other.

```
SERVERS
Server Peer   ID   State    Status     Queue  Connection Changed
-------------------------------------------------------------------------
cdr1   cdr1   1    Active   Local      0
       cdr2   2    Active   Connected  0      Apr 15 10:46:16
       cdr3   3    Active   Connected  0      Apr 15 10:46:16
       cdr4   4    Active   Connected  0      Apr 15 10:46:15
cdr2   cdr1   1    Active   Connected  0      Apr 15 10:46:16
       cdr2   2    Active   Local      0
       cdr3   3    Active   Connected  0      Apr 15 10:46:16
       cdr4   4    Active   Connected  0      Apr 15 10:46:16
cdr3   cdr1   1    Active   Connected  0      Apr 15 10:46:16
       cdr2   2    Active   Connected  0      Apr 15 10:46:16
       cdr3   3    Active   Local      0
       cdr4   4    Active   Connected  0      Apr 15 10:46:16
cdr4   cdr1   1    Active   Connected  0      Apr 15 10:46:16
       cdr2   2    Active   Connected  0      Apr 15 10:46:16
       cdr3   3    Active   Connected  0      Apr 15 10:46:16
       cdr4   4    Active   Local      0
```

In this example, each of the four servers are connected to each other.

The output of this command is similar to the output of the **cdr list server** command, except that the **cdr view server** command shows all servers in the Enterprise Replication domain, not just the servers connected to the one from which the command is run. For information about the columns in this output, see “cdr list server” on page A-103.

The cdr view sendq Command Output

The following example of the output of the **cdr view sendq** command shows information about the send queue for each server.

```
RQM SENDQ
Server    Trans.    Trans.    Trans.      Data    Memory      ACKS
          in que    in mem   spooled  in queue    in use   pending
-------------------------------------------------------------------------
cdr1         594       594         0     49896     49896         0
cdr2           0         0         0         0         0         0
cdr3           0         0         0         0         0         0
cdr4           0         0         0         0         0         0
```

In this example, only the server **cdr1** has transactions in the send queue, all of which are in memory.

The columns of the **cdr view sendq** command provide the following information in addition to the server name:

Trans. in que
The number of transactions in the send queue.

Trans. in mem
The number of transactions in the send queue that are currently in memory.

Trans. spooled
The number of transactions in the send queue that have been spooled to disk.

Data in queue
The number of bytes of data in the send queue, including both in-memory and spooled transactions.

Memory in use
The number of bytes of data in the send queue that resides in memory.

ACKS pending
The number of acknowledgments that have been received but have not yet been processed.

The cdr view rcv Command Output

The following example of the output of the **cdr view rcv** command shows information about the receive queue for each server.

```
RCV
Server Received Spooled    Memory Pending Waiting
           Txn.    Txn.    In Use    Txn.    Txn.
------------------------------------------------------------------------------
cdr1          0       0         0       0       0
cdr2        372       0    871164     372       0
cdr3        220       0     18480     220       0
cdr4          0       0         0       0       0
```

In this example, the servers **cdr2** and **cdr3** have transactions in the receive queue, all of which have been preprocessed and are in the pending state waiting to be applied.

The columns of the **cdr view rcv** command provide the following information in addition to the server name:

Received Txn.
The number of transactions in the receive queue.

Spooled Txn.
The number of transactions in the receive queue that have been spooled to disk.

Memory In Use
The size, in bytes, of the receive queue.

Pending Txn.
The number of transactions that have been preprocessed but not yet applied.

Waiting Txn.
The number of acknowledgments waiting to be sent back to the source server.

The cdr view apply Command Output

The following example of the output of the **cdr view apply** command shows how replicated data is being applied.

```
APPLY
Server   Pl  Failure       Num     Num   Apply  --Latency--  ATS  RIS
       Rate    Ratio       Run  Failed    Rate  Max    Avg.    #    #
------------------------------------------------------------------------------
```

```
cdr1      0    0.000         0       0   0.000    0   0.000    0    0
cdr2      0    0.000     10001       0   0.112    0   0.000    0    0
cdr3      0    0.000     10001       0   0.112    0   0.000    0    0
cdr4      0    0.000     10001       0   0.112    0   0.000    0    0
```

In this example, the servers **cdr2**, **cdr3**, and **cdr4** each applied 10 001 transactions.

The columns of the **cdr view apply** command provide the following information in addition to the server name:

Pl Rate
Indicates the degree of parallelism used when data is being applied. Zero indicates the highest possible rate of parallelism.

Failure Ratio
The ratio of the number of times data could not be applied in parallel because of deadlocks or lock time outs.

Num Run
The number of transactions processed.

Num Failed
The number of failed transactions because of deadlocks or lock time outs.

Apply Rate
The number of transactions that have been applied divided by the amount of time that replication has been active. The Apply Rate is equal to the Commit Rate in the **cdr view profile** command.

Max. Latency
The maximum number of seconds for processing any transaction.

Avg. Latency
The average number of seconds of the life cycle of a replicated transaction.

ATS # The number of ATS files.

RIS # The number of RIS files.

The cdr view nif Command Output

The following example of the output of the **cdr view nif** command shows the status and statistics of connections between servers.

```
NIF
Source Peer   State         Messages  Messages Messages  Transmit
                               Sent  Received  Pending      Rate
------------------------------------------------------------------------
cdr1   cdr2   Connected        24014       372        6 21371.648
       cdr3   Connected        24020        17        0 20527.105
       cdr4   Connected        24014        23        6 21925.727
cdr2   cdr1   Connected          392     24015        0 21380.879
       cdr3   Connected           14        14        0    10.857
       cdr4   Connected           14        14        0    11.227
cdr3   cdr1   Connected           17     24021        0 20310.611
       cdr2   Connected           14        14        0    10.739
       cdr4   Connected           14        14        0    11.227
cdr4   cdr1   Connected          236     24015        0 21784.225
       cdr2   Connected           14        14        0    11.101
       cdr3   Connected           14        14        0    11.101
```

In this example, all servers are connected to each other. The server **cdr1** has six messages that have not yet been sent to server **cdr2** and server **cdr4**.

The columns of the **cdr view nif** command provide the following information in addition to the source server name:

Peer The name of the server to which the source server is connected.

State The connection state. Values include:

Connected
The connection is active.

Disconnected
The connection was explicitly disconnected.

Timeout
The connection attempt has timed out, but will be reattempted.

Logic error
The connection disconnected due to an error during message transmission.

Start error
The connection disconnected due to an error while starting a thread to receive remote messages.

Admin close
Enterprise Replication was stopped by a user issuing the **cdr stop** command.

Connecting
The connection is being established.

Never Connected
The servers have never had an active connection.

Messages Sent
The number of messages sent from the source server to the target server.

Messages Received
The number of messages received by the source server from the target server.

Messages Pending
The number of messages that the source server needs to send to the target server.

Transmit Rate
The total bytes of messages sent and received by the server divided by the amount of time that Enterprise Replication has been running. Same as the Throughput field in the **cdr view profile** command.

The cdr view ats and cdr view ris Command Output

The following example of the output of the **cdr view ats** command shows that there are no ATS files in text format.

```
ATS for cdr1 - no files
--------------------------------------------------------------------------------

ATS for cdr2 - no files
--------------------------------------------------------------------------------

ATS for cdr3 - no files
--------------------------------------------------------------------------------

ATS for cdr4 - no files
```

The following example of the **cdr view ris** command shows two RIS files in text format.

```
RIS for cdr1 - no files
------------------------------------------------------------------------------

RIS for cdr2 - 1 files
Source Txn. Commit         Receive
       Time                Time
------------------------------------------------------------------------------
cdr1   08-04-15 11:56:13 | 08-04-15 11:56:14
File:ris.cdr2.cdr1.D_4.080415_11:56:14.1

Row:2 / Replicate Id: 262146 / Table: stores_demo@user.customer / DbOp:Update
CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
------------------------------------------------------------------------------

RIS for cdr3 - no files
------------------------------------------------------------------------------

RIS for cdr4 - 1 files
Source Txn. Commit         Receive
       Time                Time
------------------------------------------------------------------------------
cdr1   08-04-15 11:56:13 | 08-04-15 11:56:14
File:ris.cdr4.cdr1.D_1.080415_11:56:14.1

Row:3 / Replicate Id: 262146 / Table: stores_demo@user.customer / DbOp:Update
CDR:6 (Error: Update aborted, row does not exist in target table) / SQL:0 / ISAM:0
```

In this example, the servers **cdr2** and **cdr4** each have one RIS file.

For more information on ATS and RIS files, see Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1.

The cdr view atsdir and cdr view risdir Command Output

The **cdr view atsdir** command and **cdr view risdir** command outputs have the same format. The following example of the output of the **cdr view risdir** command shows the names of two RIS files.

```
RISDIR
Server File                                  Size              Create
       Name                                                    Time
------------------------------------------------------------------------------
cdr2   ris.cdr2.cdr1.D_4.080415_11:56:14.1    465  2008-04-15 11:56:15
cdr4   ris.cdr4.cdr1.D_1.080415_11:56:14.1    475  2008-04-15 11:56:15
```

In this example, both server **cdr2** and server **cdr4** have a single RIS file. The Size column shows the size of the file, in bytes.

Examples

The following command would display information about the send queue and the network every 10 seconds:

```
cdr view sendq nif --repeat=10
```

The following command could be used in a daemon or script that runs every five minutes to check all servers for ATS and RIS files, repair inconsistencies, and delete the processed ATS and RIS files:

```
cdr view atsdir risdir --repair --delete --repeat=300
```

Related concepts:
"Monitor Enterprise Replication" on page 9-1
Chapter 9, "Monitoring and Troubleshooting Enterprise Replication," on page 9-1
"Preparing for Role Separation (UNIX)" on page 4-22
Related reference:
"cdr list server" on page A-103
"cdr repair" on page A-122
"onstat -g ddr" on page E-6

Appendix B. Enterprise Replication configuration parameter and environment variable reference

You can use configuration parameters and environment variables to configure the behavior of Enterprise Replication.

The database server **onconfig** configuration file includes the configuration parameters that affect the behavior of Enterprise Replication. If you use both the DBSERVERNAME and DBSERVERALIASES configuration parameters, the DBSERVERNAME configuration parameter must specify the network connection and not to a shared-memory connection. For information about database server aliases, see the *IBM Informix Administrator's Guide*.

Use the CDR_ENV configuration parameter to set the environment variables that affect the behavior of Enterprise Replication.

You can view the setting of Enterprise Replication configuration parameters and environment variables with the **onstat -g cdr config** command. See "onstat -g cdr config" on page E-4.

Related concepts:

"Configuring network encryption for replication servers" on page 4-5

Related tasks:

"Dynamically Modifying Configuration Parameters for a Replication Server" on page 8-1

"Setting Configuration Parameters" on page 4-14

CDR_APPLY Configuration Parameter

Specifies the minimum and maximum number of data sync threads.

onconfig.std value
does not appear in **onconfig.std**

default value if not present in the onconfig.std
the number of CPU virtual processors (VPs), four times the number of CPU VPs

syntax CDR_APPLY *min_threads*, *max_threads*

range of values
positive integers

takes effect
when the database server is shut down and restarted.

Use the CDR_APPLY configuration parameter to specify the number of data sync threads that are dynamically allocated as needed. By default, Enterprise Replication allocates a range of one to four data sync threads for each CPU VP.

CDR_DBSPACE Configuration Parameter

Specifies the dbspace where the **syscdr** database is created.

onconfig.std value
none

units any valid dbspace

takes effect
When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_DBSPACE configuration parameter specifies the dbspace where the **syscdr** database is created.

If the CDR_DBSPACE configuration parameter is not set, then **syscdr** is created in the root dbspace.

CDR_DELAY_PURGE_DTC configuration parameter

Specifies how long to retain rows in delete tables to support the delete wins conflict resolution rule.

onconfig.std value
0

default value if not present in the onconfig
0

syntax CDR_DELAY_PURGE_DTC *timeunit*

range of values
The range of values for *time* are 0 and positive integers.

The range of values for *unit* are:

- S = seconds (Default)
- M = minutes
- H = hours
- D = days

takes effect
After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the **onmode -wf** command.

When you reset the value for a session by running the **onmode -wm** command.

By default, rows in delete tables are deleted when those rows are no longer required by the timestamp conflict resolution rule. If you want to perform time stamp repair and your replicates use the delete wins conflict resolution rule, set the CDR_DELAY_PURGE_DTC configuration parameter to the maximum age of modifications to rows that are being actively updated. The longer you retain rows in delete tables, the more accurate time stamp repairs are, but the more disk space the delete tables consume.

Tip: Right before you enable a disabled server, dynamically update the CDR_DELAY_PURGE_DTC configuration parameter to set it to a value slightly greater than the time that the server was disabled plus the amount of time a repair takes.

Related concepts:

"Repair inconsistencies by time stamp" on page 8-20

"Delete Wins Conflict Resolution Rule" on page 3-12

Related reference:

onmode -wf, -wm: Dynamically change certain configuration parameters (Administrator's Reference)

CDR_DSLOCKWAIT Configuration Parameter

Specifies the number of seconds the data sync component waits for the database locks to be released.

onconfig.std value
5

units seconds

takes effect
When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_DSLOCKWAIT configuration parameter specifies the number of seconds the data sync component waits for database locks to be released. The CDR_DSLOCKWAIT parameter behaves similarly to the SET LOCK MODE statement. Although 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 data sync due to lock timeouts like this, you might want to increase the value of this parameter.

CDR_ENV Configuration Parameter

Sets the Enterprise Replication environment variables CDR_ALARMS, CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, or CDR_RMSCALEFACT.

Important: Use the CDR_LOGDELTA, CDR_PERFLOG, CDR_ROUTER, and CDR_RMSCALEFACT environment variables only if instructed to do so by IBM Support.

units Enterprise Replication environment variable name and value, separated by an equal sign

takes effect
When the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command

The `onconfig` file can contain multiple entries for the CDR_ENV environment variable. You can specify only one environment variable per CDR_ENV entry.

The following line in the onconfig file sets the CDR_ALARMS environment variable to add event alarm 51 to the event alarms that are enabled by default:

```
CDR_ENV CDR_ALARMS=30-39,47,48,50,51,71,73-75
```

When you update the CDR_ALARMS environment variable in the onconfig file, you must list all the Enterprise Replication event alarms that you want to be enabled.

The following lines in the onconfig file set the CDR_LOGDELTA environment variable to 30 and the CDR_ROUTER environment variable to 1:

```
CDR_ENV CDR_LOGDELTA=30
CDR_ENV CDR_ROUTER=1
```

Related tasks:

"Enabling or Disabling Enterprise Replication Event Alarms" on page 9-40

CDR_EVALTHREADS Configuration Parameter

Specifies the number of group evaluator threads to create when Enterprise Replication starts, and enables parallelism.

onconfig.std value
: 1,2

units evaluator thread instances

range of values
: first value: positive integer representing the number of evaluator threads per CPU VP

: second value: 0 or a positive integer representing the additional number of evaluator threads

takes effect
: When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

Enterprise Replication evaluates the images of a row in parallel to assure high performance. Figure B-1 on page B-5 illustrates how Enterprise Replication uses parallel processing to evaluate transactions for replication.

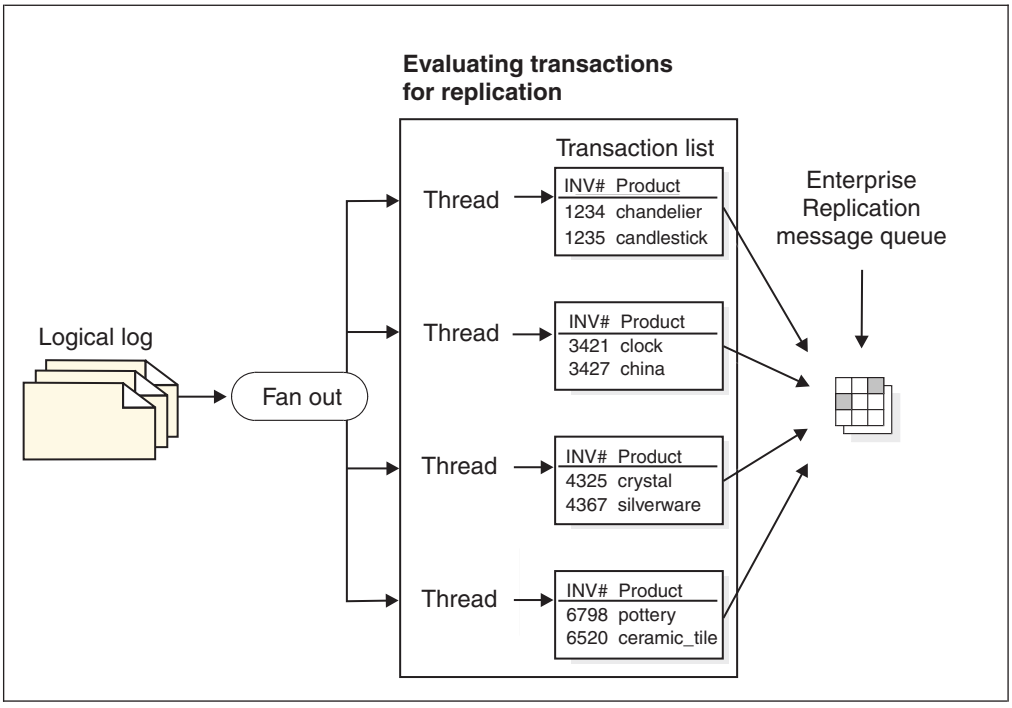


Figure B-1. Processing in Parallel for High Performance

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

Attention: Do not configure the total number of evaluator threads to be smaller than the number of CPU VPs in the system.

CDR_LOG_LAG_ACTION Configuration Parameter

Specifies how Enterprise Replication responds to a potential log wrap situation.

onconfig.std value
ddrblock

default value if not present in the onconfig file
ddrblock

separators
+

range of values

One of the following combinations of options:

- logstage+dlog+ignore
- logstage+dlog+ddrblock
- logstage+dlog+shutdown
- logstage+ignore
- logstage+ddrblock
- logstage+shutdown
- dlog+logstage+ignore
- dlog+logstage+shutdown
- dlog+logstage+ddrblock
- dlog+ignore
- dlog+ddrblock
- dlog+shutdown
- ignore
- ddrblock
- shutdown
- logstage
- dlog
- logstage+dlog
- dlog+logstage

takes effect

After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the **onmode -wf** command.

When you reset the value for a session by running the **onmode -wm** command.

Use the CDR_LOG_LAG_ACTION configuration parameter to specify one or more actions, in priority order, that Enterprise Replication takes during a potential log wrap situation.

Table B-1. CDR_LOG_LAG_ACTION options

Option	Description
logstage	Enables compressed logical log staging. The following configuration parameters must also be set: • The LOG_STAGING_DIR configuration parameter must be set to a directory. The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user informix, must belong to group informix, and must not have public read, write, or execute permission. • The CDR_LOG_STAGING_MAXSIZE configuration parameter must be set to a positive number. Log files are staged in the directory specified by the LOG_STAGING_DIR configuration parameter, until the maximum size specified by the CDR_LOG_STAGING_MAXSIZE configuration parameter is reached. The staged log files are deleted after advancing the log replay position. If the amount of disk space specified by the CDR_LOG_STAGING_MAXSIZE configuration parameter is exceeded, event alarm ID 30 is raised with alarm ID 30005 Alarm class message: DDR Subsystem notification Alarm-specific message: CDR DDR: Log staging disk space usage reached its allowed configured maximum size *size* (KB). Temporarily disabling log staging. If log staging is configured, Enterprise Replication monitors the log lag state and stages log files even when Enterprise Replication is inactive.
dlog	Enables the dynamic addition of logical logs. The following configuration parameters must be set: • The CDR_MAX_DYNAMIC_LOGS configuration parameter must be set to -1 or a positive number. • The DYNAMIC_LOGS configuration parameter must be set to 2.
ignore	Ignore the potential for log wrapping. The Enterprise Replication replay position might be overrun. If the replay position is overrun, event alarm 30 is raised. Restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data. The **ignore** option must be the only or the last option. If the snoopy log position overrun is detected, Enterprise Replication shuts down with alarm class ID 47 and unique ID 47005. Alarm class message: CDR is shutting down due to internal error: failure Alarm-specific message: CDR is shutting down due to internal error: Internal shutdown
ddrblock	(Default) Block client applications update activity. The **ddrblock** option must be the only or the last option.

Table B-1. CDR_LOG_LAG_ACTION options (continued)

Option	Description
shutdown	Shut down Enterprise Replication on the affected server. If replay position overrun is detected, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data. If the replay position was not overrun, restart Enterprise Replication using the **cdr start** command; there is no need to synchronize the data. If replay position overrun is detected and the **cdr start** command fails with error code 214 and raises alarm class ID 75, restart Enterprise Replication using the **cdr cleanstart** command and synchronize the data. The **shutdown** option must be the only or the last option. If a log lag state is detected, Enterprise Replication is shut down and alarm ID 47 is raised with unique ID 47006 Alarm class message: CDR is shutting down due to internal error: log lag state Alarm-specific message: CDR DDR: Shutting down ER to avoid a DDRBLOCK situation.

Staged log file format

Enterprise Replication creates a directory named: `ifmxddrlog_<SERVERNUM>` in the directory specified by the LOG_STAGING_DIR configuration parameter. Log file names are in the following format:

```
ifmxERDDRBLOCKUniqueLog_<lf_used>_<loguniqueid>.dat
```

Enterprise Replication also creates an empty token file for each staged log file. The token file is used to detect log files that are only partially written. If a token file is not found then Enterprise Replication treats the staged log file as partially written log file and deletes it. The token log file format is:

```
ifmxERDDRBLOCKUniqueLog_<lf_used>_<loguniqueid>
```

Transferring log files to a high-availability cluster secondary server when using ER

If your configuration consists of an HDR, RSS, or SDS secondary server configured as an Enterprise Replication node, transfer staged log files to the secondary server using the alarm program script. The staged log files are required by Enterprise Replication in case the primary server in a high-availability cluster fails and a secondary server takes over the role of the primary server.

Enterprise Replication raises alarm class ID 30 and unique ID 30006 when a log is staged to the log staging directory. Enterprise Replication raises alarm class ID 30 and unique ID 30007 after deleting a staged log file. Using these alarms, you can automate the transfer of staged log files to the high-availability cluster secondary server using the alarm program script.

Ensure that the directory under the directory specified the LOG_STAGING_DIR configuration parameter exists and is named using the format `ifmxddrlog_<SERVERNUM>`. The script copies the staged log files to `ifmxddrlog_<SERVERNUM>` and creates a token log file after copying the staged log file.

Example

Suppose that you want Enterprise Replication to handle potential log wrap situations by first staging compressed logs until they reach 1 MB in size, then dynamically add up to two logical logs, and then block user transactions. Set the following configuration parameters:

```
CDR_LOG_LAG_ACTION      logstage+dlog+ddrblock
LOG_STAGING_DIR         $INFORMIXDIR/tmp
CDR_LOG_STAGING_MAXSIZE 1MB
CDR_MAX_DYNAMIC_LOGS    2
DYNAMIC_LOGS            2
```

Related concepts:

"Handle potential log wrapping" on page 9-15

Related reference:

LOG_STAGING_DIR configuration parameter (Administrator's Reference)

onmode -wf, -wm: Dynamically change certain configuration parameters (Administrator's Reference)

CDR_LOG_STAGING_MAXSIZE Configuration Parameter

Specifies the maximum amount of space that Enterprise Replication uses to stage compressed log files in the directory specified by the LOG_STAGING_DIR configuration parameter.

onconfig.std value
none

default value if not present in the onconfig file
none

syntax CDR_LOG_STAGING_MAXSIZE *sizeunit*

range of values
The range of values for *size* is positive integers.

The range of value for *unit* is:

- KB (default)
- MB
- GB
- TB

takes effect
After you edit your `onconfig` file and restart the database server.

When you reset the value dynamically in your `onconfig` file by running the **onmode -wf** command.

When you reset the value for a session by running the **onmode -wm** command.

Use the CDR_LOG_STAGING_MAXSIZE configuration parameter to limit the size of the log staging directory. Logs are staged if all of the following conditions are true:

- Enterprise Replication detects a potential for log wrapping.
- The CDR_LOG_LAG_ACTION configuration parameter setting includes the **logstage** option.
- The LOG_STAGING_DIR configuration parameter is set.

The directory specified by the LOG_STAGING_DIR configuration parameter must be secure. The directory must be owned by user informix, must belong to group informix, and must not have public read, write, or execute permission.

When the contents of the staging directory reaches the maximum allowed size, Enterprise Replication stops staging log files. Enterprise Replication stops staging files only at a log file boundary; that is, a file is not staged in the middle of a log file.

Example

Suppose that you want Enterprise Replication to handle potential log wrap situations by staging compressed logs until the staging directory reached 100 KB, you would set the following configuration parameters:

```
CDR_LOG_STAGING_MAXSIZE 100
CDR_LOG_LAG_ACTION      logstage
LOG_STAGING_DIR         $INFORMIXDIR/tmp
```

Related concepts:

"Handle potential log wrapping" on page 9-15

Related reference:

LOG_STAGING_DIR configuration parameter (Administrator's Reference)

onmode -wf, -wm: Dynamically change certain configuration parameters (Administrator's Reference)

CDR_MAX_DYNAMIC_LOGS Configuration Parameter

Specifies the number of dynamic log file requests that Enterprise Replication can make in one server session.

onconfig.std value

`0`

range of values

- `-1` add dynamic log files indefinitely
- `0` disable dynamic log addition
- `>0` number of dynamic logs that can be added

takes effect

when the database server is shut down and restarted, and the DYNAMIC_LOGS configuration parameter is set to 2 or when the **cdr change onconfig** command is used. For more information on the DYNAMIC_LOGS configuration parameter, see the *IBM Informix Administrator's Reference*.

The CDR_MAX_DYNAMIC_LOGS configuration parameter specifies the number of dynamic log file requests that Enterprise Replication can make in one server session. The DYNAMIC_LOGS configuration parameter must be set to 2.

Related concepts:

"Handle potential log wrapping" on page 9-15

Related reference:

DYNAMIC_LOGS configuration parameter (Administrator's Reference)

CDR_NIFCOMPRESS Configuration Parameter

Specifies the level of compression the database server uses before sending data from the source database server to the target database server.

onconfig.std 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 the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

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.

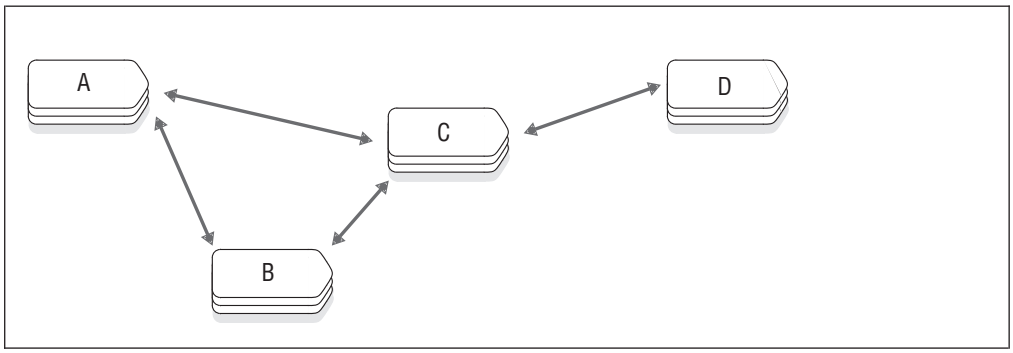


Figure B-2. Database Servers with Different Compression Levels

Important: Do not disable NIF compression if the network link performs compression in hardware.

CDR_QDATA_SBSPACE Configuration Parameter

Specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data.

onconfig.std value
none

separators
comma

range of values
up to 128 characters for each sbspace name; up to 32 sbspace names. Use a comma to separate each name in the list. At least one sbspace name must be specified.

takes effect
when the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

The CDR_QDATA_SBSPACE configuration parameter specifies the list of up to 32 names of sbspaces that Enterprise Replication uses to store spooled transaction row data. Enterprise Replication creates one smart large object per transaction. The sbspaces must be used only for Enterprise Replication. If CDR_QDATA_SBSPACE is configured for multiple sbspaces, then Enterprise Replication uses all appropriate sbspaces in round-robin order.

Important: You must set the CDR_QDATA_SBSPACE configuration parameter and create the sbspaces specified by CDR_QDATA_SBSPACE before defining a server for replication. If the configuration parameter is not set in ONCONFIG or the sbspace names specified by CDR_QDATA_SBSPACE are invalid, Enterprise

Replication fails to define the server. For more information, see “Row Data sbspaces” on page 4-10 and “Defining Replication Servers” on page 6-1.

Warning: Do not change the value of CDR_QDATA_SBSPACE while Enterprise Replication is running.

Related tasks:

“Monitoring Disk Usage for Send and Receive Queue Spool” on page 9-16

Related reference:

“cdr start sec2er” on page A-137

CDR_QUEUEMEM Configuration Parameter

Specifies the maximum amount of memory that is used for the send and receive queues.

onconfig.std value
`4096`

units kilobytes

range of values
From `500` through `4194304`

takes effect
When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_QUEUEMEM configuration parameter specifies the maximum amount of memory that the send and receive queues use for transaction headers and for transaction data. The total size of the transaction headers and transaction data in a send or receive queue could be up to twice the size of that value of CDR_QUEUEMEM. 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.

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_SERIAL Configuration Parameter

Enables control over generating values for serial columns in tables defined for replication.

onconfig.std value
`0`

units *delta*, *offset*

range of values
`0` disable control of serial column value generation

`two positive integers separated by a comma` enable control of serial column value generation

takes effect
When the database server is shut down and restarted or immediately after the **cdr change onconfig** command is used

The CDR_SERIAL configuration parameter enables control over generating values for SERIAL, SERIAL8, and BIGSERIAL primary key columns in replicated tables so that no conflicting values are generated across multiple Enterprise Replication servers. CDR_SERIAL is necessary if the serial column is the primary key column and no other primary key column, such as a server ID, guarantees the uniqueness of the primary key. 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 values.

The format is:

```
CDR_SERIAL delta,offset
```

where:

delta Determines the incremental size of the serial column values. This value must be the same on all replication servers and must be at least the number of expected servers in the Enterprise Replication domain.

offset Determines the offset of the serial value that will be generated. This value must be different on all replication servers and must be between 0 and one less than the value of *delta*, inclusive.

For example, suppose you have two primary servers, **g_usa** and **g_japan**, and one read-only target server, **g_italy**. You plan to add three additional servers in the future. You might set CDR_SERIAL to the values shown in the following table.

Table B-2. CDR_SERIAL Example Settings and Results

Server	Example CDR_SERIAL Value	Resulting Values for the Serial Column
g_usa	5,0	5, 10, 15, 20, 25, and so on
g_japan	5,1	6, 11, 16, 21, 26, and so on
g_italy	0	no local inserts into the serial column

The CDR_SERIAL setting of `5,2`, `5,3`, and `5,4` are reserved for the future servers.

If you need to add more servers than the *delta* value of CDR_SERIAL, you must reset CDR_SERIAL on all servers simultaneously and ensure that the serial values on the new servers are unique.

For more information on using serial columns as primary keys, see "Serial Data Types and Primary Keys" on page 2-7.

CDR_SUPPRESS_ATSRISWARN Configuration Parameter

Specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files.

onconfig.std value
none

units numbers or hyphen-separated ranges of numbers

separator
commas

takes effect
when the database server is shut down and restarted or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

The CDR_SUPPRESS_ATSRISWARN configuration parameter specifies the data sync error and warning code numbers to be suppressed in ATS and RIS files. For example, you can set CDR_SUPPRESS_ATSRISWARN to 2-5, 7 to suppress the generation of error and warning messages 2, 3, 4, 5, and 7. For a list of error and message numbers see Appendix J, "Data Sync Warning and Error Messages," on page J-1.

ENCRYPT_CDR Configuration Parameter

Use the ENCRYPT_CDR configuration parameter to set the level of encryption for Enterprise Replication.

onconfig.std value
ENCRYPT_CDR 0

values 0 = Default. Do not encrypt.

1 = Encrypt when possible. Encryption is used for Enterprise Replication transactions only when the database server being connected to also supports encryption.

2 = Always encrypt. Only connections to encrypted database servers are allowed.

takes effect
After you edit your `onconfig` file and restart the database server.

After you run the **cdr change onconfig** command.

Usage

If you enable encryption with the ENCRYPT_CDR configuration parameter, you must also set the ENCRYPT_MAC, ENCRYPT_MACFILE, ENCRYPT_SWITCH, and ENCRYPT_CIPHERS configuration parameter to configure encryption.

If you use both encryption and compression (by setting the CDR_NIFCOMPRESS configuration parameter), then compression occurs before encryption.

Related reference:

ENCRYPT_CIPHERS Configuration Parameter (Administrator's Reference)

ENCRYPT_MACFILE configuration parameter (Administrator's Reference)

ENCRYPT_SWITCH configuration parameter (Administrator's Reference)

ENCRYPT_MAC Configuration Parameter (Administrator's Reference)

CDR_ALARMS Environment Variable

Enables Enterprise Replication event alarms.

default value
30-39,47,48,50,71,73-75

range of values
integers in these ranges: 30-39, 47-71, 73-75

separator
comma (,) to separate individual numbers or hyphen (-) to separate a range of numbers

takes effect
When the database server is shut down and restarted.

Set the CDR_ALARMS environment variable to the Enterprise Replication event alarms that you want to receive. Enterprise Replication event alarms that are not set by CDR_ALARMS are disabled.

Use the CDR_ENV configuration parameter to set this environment variable in the `onconfig` file.

Related tasks:

"Enabling or Disabling Enterprise Replication Event Alarms" on page 9-40

CDR_ATSRISNAME_DELIM Environment Variable

Specifies the delimiter to use to separate the parts of the time portion of ATS and RIS file names that are in text format.

default value
On UNIX: a colon (**:**)

On Windows: a period (**.**)

range of values
a single character

takes effect
when Enterprise Replication is initialized

ATS and RIS files in XML format always use a period (.) as the delimiter.

For example, the default file name for an ATS file in text format on UNIX might look like this: **ats.g_beijing.g_amsterdam.D_2.000529_23:27:16.6**. If CDR_ATSRISNAME_DELIM is set to a period (.), then the same file name would look like this: **ats.g_beijing.g_amsterdam.D_2.000529_23.27.16.6**.

Related concepts:

"ATS and RIS File Names" on page 9-5

CDR_DISABLE_SPOOL Environment Variable

Controls the generation of ATS and RIS files.

default value
`0`

range of values
`0` Allow ATS and RIS file generation

1 Prevent ATS and RIS file generation

takes effect
when Enterprise Replication is initialized

The CDR_DISABLE_SPOOL environment variable controls whether ATS and RIS files are generated. Set CDR_DISABLE_SPOOL to 1 if you do not want ATS or RIS files to be generated under any circumstances.

Related concepts:

"Failed Transaction (ATS and RIS) Files" on page 9-3

Related tasks:

"Disabling ATS and RIS File Generation" on page 9-13

CDR_LOGDELTA Environment Variable

Determines when the send and receive queues are spooled to disk as a percentage of the logical log size.

default value
`30`

range of values
positive numbers

takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_LOGDELTA environment variable determines when the send and receive queues are spooled to disk as a percentage of the logical log size. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see "CDR_ENV Configuration Parameter" on page B-3.

Important: Do not use the CDR_LOGDELTA environment variable unless instructed to do so by Technical Support.

CDR_PERFLOG Environment Variable

Enables queue tracing.

default value
`0`

range of values
positive number

takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_PERFLOG environment variable enables queue tracing. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see "CDR_ENV Configuration Parameter" on page B-3.

Important: Do not use the CDR_PERFLOG environment variable unless instructed to do so by Technical Support.

CDR_RMSCALEFACT Environment Variable

Sets the number of data sync threads started for each CPU VP.

default value
4

range of values
positive number

takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

The CDR_RMSCALEFACT environment variable sets the number of data sync threads started for each CPU VP. Specifying a large number of threads can result in wasted resources. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see "CDR_ENV Configuration Parameter" on page B-3.

Important: Do not use the CDR_RMSCALEFACT environment variable unless instructed to do so by Support.

CDR_ROUTER Environment Variable

Disables intermediate acknowledgments of transactions in the hierarchical topologies.

default value
0

range of values
any number

takes effect
when Enterprise Replication is initialized or immediately after the **cdr change onconfig** command is used

When set to 1, the CDR_ROUTER environment variable disables intermediate acknowledgments of transactions in hierarchical topologies. The normal behavior for intermediate servers is to send acknowledgments if they receive an acknowledgment from the next server in the replication tree (can be a leaf server) or if the transaction is stored in the local queue. Use the CDR_ENV configuration parameter to set this environment variable. For more information, see "CDR_ENV Configuration Parameter" on page B-3.

If CDR_ROUTER is set at the hub server, an acknowledgment will be sent only if the hub server receives acknowledgment from all of its leaf servers. Transactions will not be acknowledged even if they are stored in the local queue of the hub server.

If CDR_ROUTER is not set at hub server, the hub server will send an acknowledgment if the transaction is stored in the local queue at the hub server or if the hub server received acknowledgment from all of its leaf servers.

Important: Do not use the CDR_ROUTER environment variable unless instructed to do so by Technical Support.

CDRSITES_10X Environment Variable

Works around a malfunction in version reporting for fix pack versions of 10.00 servers.

units ***cdrIDs***, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values
positive numbers

takes effect
when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_10X** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve Versions 10.00.xC1 or 10.00.xC3 servers, the NIF does not properly report its version when it responds to a new server with a fix pack version of 10.00.xC4 or later. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server.

To prevent this malfunction, if you have Version 10.00.xC1 or 10.00.xC3 servers in your Enterprise Replication environment, set the **CDRSITES_10X** environment variable with the CDR_ENV configuration parameter for these servers.

Note: You can only set the **CDRSITES_10X** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_10X** as a standard environment variable.

The ***cdrID*** is the unique identifier for the database server in the Options field of the SQLHOSTS file (**i = *unique_ID***).

For example, suppose that you have 5 database servers, Version 10.00.xC1, whose ***cdrID*** values range from 2 through 10 (***cdrID*** = 2, 3, 8, 9, and 10).

If you upgrade database server ***cdrID*** 8 to Version 10.00.xC4, you must set the **CDRSITES_10X** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.00.xC4 database server online:

```
CDR_ENV CDRSITES_10x=2,3,9,10
```

CDRSITES_731 Environment Variable

Works around a malfunction in version reporting for post-7.3x, 7.20x, or 7.24x version servers.

units ***cdrIDs***, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values
positive numbers

takes effect
when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_731** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command

- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve post 7.3x, 7.20x, or 7.24x servers, the NIF does not properly report its version when it responds to a new server. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server. If a 10.0, 9.40, or 9.30 server tries to synchronize with a 7.3x, 7.20x, or 7.24x server, the older server responds to the 10.0, 9.40, or 9.30 server that it is a 10.0, 9.40, or 9.30 server and will subsequently fail.

To prevent this malfunction, if you have Version 7.3x, 7.20x, or 7.24x servers in your Enterprise Replication environment, set the **CDRSITES_731** environment variable with the CDR_ENV configuration parameter for these servers.

Note: You can only set the **CDRSITES_731** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_731** as a standard environment variable.

For example, suppose that you have 5 database servers, Version 7x servers whose ***cdrID*** values range from 1 through 7 (***cdrID*** = 1, 4, 5, 6, and 7).

If you upgrade database server ***cdrID*** 6 to Version 10.0, 9.40, or 9.30, you must set the **CDRSITES_731** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.0, 9.40, or 9.30 database server online:

```
CDR_ENV CDRSITES_731=1,4,5,7
```

CDRSITES_92X Environment Variable

Works around a malfunction in version reporting for 9.21 or 9.20 servers.

units ***cdrIDs***, which are the unique identifiers for the database server in the Options field of the SQLHOSTS file (**i =*unique_ID***)

range of values
positive numbers

takes effect
when Enterprise Replication is initialized and the CDR_ENV configuration parameter has a value for **CDRSITES_92X** in the ONCONFIG file, or immediately for the following actions:

- Adding a value using the **cdr add onconfig** command
- Removing a value using the **cdr remove onconfig** command
- Changing a value using the **cdr change onconfig** command

In mixed-version Enterprise Replication environments that involve 9.21 or 9.20 servers, the NIF does not properly report its version when it responds to a new server. When a new server sends an initial protocol message to a sync server, the sync server, instead of properly giving its version, gives back the version of the new server. If a 10.0/9.40/9.30 server tries to synchronize with a 9.21 or 9.20 server, the older server responds to the 10.0, 9.40, or 9.30 server that it is a 10.0, 9.40, or 9.30 server and will subsequently fail.

To prevent this malfunction, if you have Version 9.21 or 9.20 servers in your Enterprise Replication environment, set the **CDRSITES_92X** environment variable

Note: You can only set the **CDRSITES_92X** environment variable by using the CDR_ENV configuration parameter. You cannot set **CDRSITES_92X** as a standard environment variable.

For example, suppose that you have 5 database servers, Version 9.21 or 9.20 whose ***cdrID*** values range from 2 through 10 *(cdrIDs* = 2, 3, 8, 9, and 10).

If you upgrade database server ***cdrID*** 8 to Version 10.0, 9.40, or 9.30, you must set the **CDRSITES_92X** environment variable for the other server ***cdrIDs*** by setting the CDR_ENV configuration parameter in the ONCONFIG file before bringing the Version 10.0, 9.40, or 9.30 database server online:

```
CDR_ENV CDRSITES_92x=2,3,9,10
```

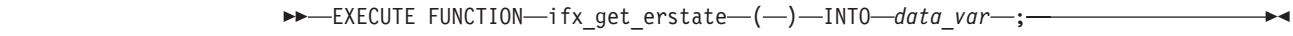
Appendix C. Grid routines

Grid routines are used to create and maintain the grid and to administer servers in the grid by propagating commands from a source server to all other servers in the grid.

ifx_get_erstate() function

The **ifx_get_erstate()** function indicates whether replication is enabled for the transaction in which it is run.

Syntax



Element	Purpose	Restriction
data_var	Variable to receive the value that the function returns	

Usage

Use the **ifx_get_erstate()** function to obtain the state of replication within a transaction. You can use the state information saved in the variable as input to the **ifx_set_erstate()** procedure.

Return value

A return value of 1 indicates that the current transaction is replicating data.

A return value of 0 indicates that the current transaction is not replicating data.

Example

The following example obtains the replication state and stores it in the **curstate** variable:

```
EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
```

Related concepts:

"Recapture replicated transactions" on page 8-28

Related tasks:

"Enabling replication within a grid transaction" on page 7-14

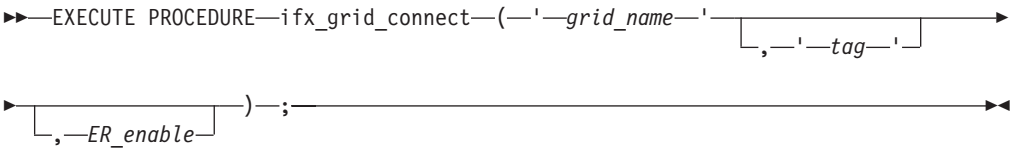
Related reference:

"ifx_set_erstate() procedure" on page C-10

ifx_grid_connect() procedure

The **ifx_grid_connect()** procedure opens a connection to the grid for running data definition language (DDL) SQL statements and routines on a source server and propagating them to the other servers in the grid.

Syntax



Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
ER_enable	Enable or disable the creation of a replicate and replicate set and starting replication for any tables created while the connection to the grid is open. Optionally suppress any errors that could be raised when the procedure is run.	Valid values are: • 0 = Default. Replication is disabled. • 1 = Replication is enabled. • 2 = Replication is disabled and errors are suppressed. • 3 = Replication is enabled and errors are suppressed.
tag	A character string to identify grid operations.	

Usage

Use the **ifx_grid_connect()** procedure to propagate the DDL SQL statements and routines following it to all the servers in the grid. Use the **ifx_grid_disconnect()** procedure to disable grid propagation. The **ifx_grid_connect()** procedure does not propagate data manipulation language statements through the grid.

If the databases on your replication servers have different schemas or data, a DDL statement run through a grid could have different results on each server. In a replication system, when you run a statement locally, the results are replicated to the other replication servers. When you run a statement through a grid, that statement is simultaneously run on each server.

If you run a DDL statement through a grid to create a database object and that object already exists on a server in the grid, you receive an SQL error that the statement could not be completed on that server.

If you set the *ER_enable* option to 1 when you create a table through the grid, a replicate is created that contains the newly created table with all the servers in the grid as participants. The replicate belongs to a replicate set that has the same name as the grid. When you create a replicated table through the grid, the ERKEY shadow columns are added automatically.

If you run the **ifx_grid_connect()** procedure automatically as part of the **sysdbopen()** procedure, set the value of the *ER_enable* option to 2 or 3 to suppress any errors that could result from running the **ifx_grid_connect()** procedure. Errors that occur while the **sysdbopen()** procedure is running would result in preventing the session from accessing the database.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr enable grid** command.

You cannot use the *@servername* syntax while connected to the grid.

You cannot drop a replicated column through a grid. To drop a replicated column, you must manually remaster the replicate and then drop the column.

You cannot rename a replicated database. You must manually rename the database on each participant server.

You must connect to a database before running the **ifx_grid_connect()** function. If you are planning to create a database, you can connect to the **sysmaster** database.

Example

Example 1: Create a table

In the following example, grid propagation of SQL statements is enabled, a table is created on all servers in the grid, and then grid propagation is disabled:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1');

CREATE TABLE special_offers(
    offer_description varchar(255),
    offer_startdate date,
    offer_enddate date,
    offer_rules  lvarchar);

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Example 2: Create a replicated table

The following example creates the same table as the previous example, but also creates a replicate that uses time stamp conflict resolution:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

CREATE TABLE special_offers(
   offer_description varchar(255),
    offer_startdate date,
    offer_enddate date,
    offer_rules  lvarchar)
    WITH CRCOLS;

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Example 3: Alter a replicated table

The following example alters the **special_offers** table to add a new column and remasters the replicate on all participants that are members of the grid:

```
database sales;

EXECUTE PROCEDURE ifx_grid_connect('grid1', 1);

ALTER TABLE special_offers ADD (
   offer_exceptions varchar(255));

EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related tasks:

"Propagating database object changes" on page 7-10

"Creating replicated tables through a grid" on page 7-10

"Altering replicated tables through a grid" on page 7-11

Related reference:

"ifx_grid_disconnect() procedure"

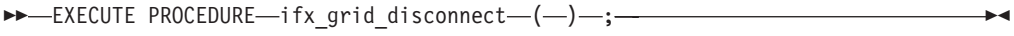
"cdr enable grid" on page A-89

"cdr delete grid" on page A-78

ifx_grid_disconnect() procedure

The **ifx_grid_disconnect()** procedure closes a connection to the grid.

Syntax



Usage

Use the **ifx_grid_disconnect()** procedure to disable the propagation of DDL statements and commands to servers in the grid, which was enabled by the **ifx_grid_connect()** procedure. If you do not use the **ifx_grid_disconnect()** procedure, propagation through the grid is stopped when the database is closed or the connection is closed.

You must run this routine as an authorized user on an authorized grid server, as specified by the **cdr grid enable** command.

Example

The following example shows how to close a connection to the grid after opening a connection:

```
EXECUTE PROCEDURE ifx_grid_connect('grid1');
EXECUTE PROCEDURE ifx_grid_disconnect();
```

Related tasks:

"Propagating database object changes" on page 7-10

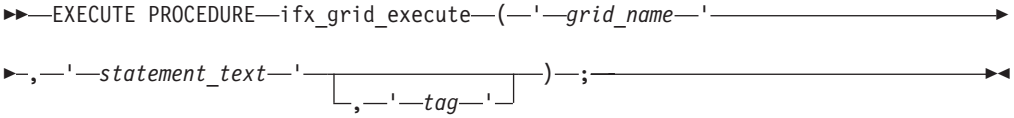
Related reference:

"ifx_grid_connect() procedure" on page C-1

ifx_grid_execute() procedure

The **ifx_grid_execute()** procedure propagates the execution of a routine or data manipulation language (DML) SQL statement to all servers in the grid.

Syntax



Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
statement_text	The text format of the routine or SQL statement to be run.	Bound data items cannot be included in procedure text.
tag	A character string to identify grid operations.	

Usage

Use the **ifx_grid_execute()** procedure to run a routine or DML SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. The output of the routine, if any, is not returned to the client application. The results of routines or statements performed within the context of the **ifx_grid_execute()** procedure are not replicated. The **ifx_grid_execute()** procedure effectively runs routines and statements with a BEGIN WORK WITHOUT REPLICATION statement. Do not use the **ifx_grid_execute()** procedure to populate tables that are already involved in replication. Although you can use the **ifx_grid_execute()** procedure to run a DML statement, for example, to delete a large number of rows from a table, in general you should use Enterprise Replication to replicate changes to replicated data.

You cannot run the **ifx_grid_execute()** procedure from within a transaction. When you run SQL administration API commands from the **ifx_grid_execute()** procedure, you must use double quotation marks around the SQL administration API function arguments and single quotation marks around the **ifx_grid_execute()** procedure arguments.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example, run from the **sysadmin** database, uses an SQL administration API command to create a dbspace on every server in the grid:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1',
  'admin("create dbspace", "dbspace3",
  "$INFORMIXDIR/WORK/dbspace3", "500 M")');
```

The following example drops the logical logs from the chunk number 3 from all the servers in the grid:

```
EXECUTE PROCEDURE ifx_grid_execute('grid1', 'SELECT task("drop log", number) FROM
     sysmaster:syslogfil where chunk = 3;');
```

Related tasks:

"Administering servers in the grid with the SQL administration API" on page 7-8

"Propagating updates to data" on page 7-12

Related reference:

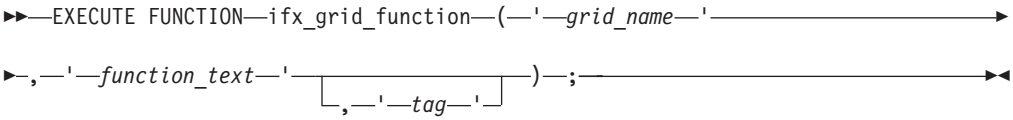
"ifx_grid_procedure() procedure" on page C-7

"ifx_grid_function() function"

ifx_grid_function() function

The **ifx_grid_function()** function propagates the execution of a function to all servers in the grid.

Syntax



Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
function_text	The text format of the function to be run.	
tag	A character string to identify grid operations.	

Usage

Use the **ifx_grid_function()** function to run a function or SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. The output of the function is returned to the client application as an LVARCHAR data type with comma-delimited text. You can also view the output with the **cdr list grid** command with the **--verbose** option. You cannot run the **ifx_grid_function()** function from within a transaction.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example runs a function named **load_function()**:

```
EXECUTE FUNCTION ifx_grid_function('grid1', 'load_function(2000)');
```

Related tasks:

"Administering servers in the grid with the SQL administration API" on page 7-8

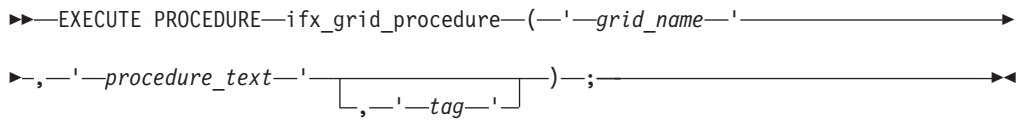
Related reference:

"ifx_grid_execute() procedure" on page C-4

ifx_grid_procedure() procedure

The **ifx_grid_procedure()** procedure propagates the execution of a procedure to all servers in the grid.

Syntax



Element	Purpose	Restrictions	Syntax
grid_name	Name of the grid.	Must be the name of an existing grid.	
procedure_text	The text format of the procedure to be run.	Bound data items cannot be included.	
tag	A character string to identify grid operations.		

Usage

Use the **ifx_grid_procedure()** procedure to run a procedure or SQL statement on a source server and propagate it so that it is also run on the other servers in the grid. You cannot run the **ifx_grid_procedure()** procedure from within a transaction.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example runs a procedure named **myloadprocedure()**:

```
EXECUTE PROCEDURE ifx_grid_procedure('grid1',
 'myloadprocedure(2000)', 'mytag');
```

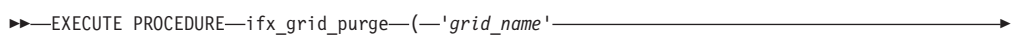
Related reference:

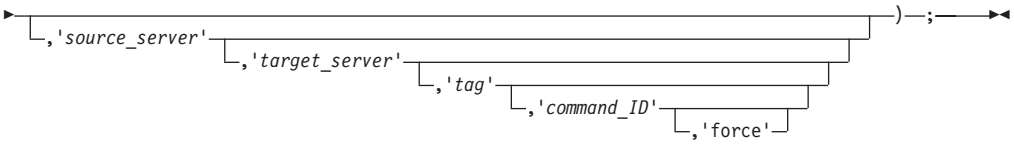
"ifx_grid_execute() procedure" on page C-4

ifx_grid_purge() procedure

The **ifx_grid_purge()** procedure deletes metadata about commands that have been run through the grid.

Syntax





Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
command_ID	One or more ID numbers of the command to purge.	Separate multiple ID numbers with a comma or specify a range with a hyphen (-). Can be NULL.
source_server	The replication server on which the routine originated.	Can be NULL.
tag	A character string identifying the grid operations to purge.	Must be an existing tag. Can be NULL.
target_server	The replication server on which the routine was run.	Can be NULL.

Usage

Use the **ifx_grid_purge()** procedure to delete the history of commands successfully run from the grid. Accumulated command history can significantly increase the size of the **syscdr** database.

Use the **force** argument to delete the history of all commands, including those that failed.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example deletes the history for commands that ran successfully:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1');
```

The following example deletes the history for commands, including those that failed:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', NULL, NULL, NULL, NULL, 'force');
```

The following example deletes the command history with the ID of 21 that originated server **cdr1** and ran on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', 'cdr1', 'cdr4', NULL, '21');
```

The following example deletes all commands that ran successfully on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_purge('grid1', NULL, 'cdr4');
```

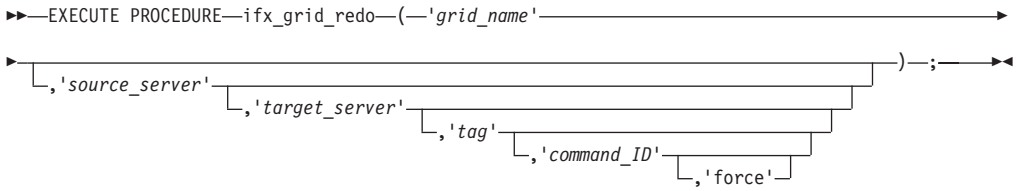
Related tasks:

"Maintaining the grid" on page 7-5

ifx_grid_redo() procedure

The **ifx_grid_redo()** procedure reruns commands that were run through the grid and failed on one or more servers in the grid.

Syntax



Element	Purpose	Restrictions
grid_name	Name of the grid.	Must be the name of an existing grid.
command_ID	One or more ID numbers of the command to rerun on the grid.	Separate multiple ID numbers with a comma or specify a range with a hyphen (-). Can be NULL.
source_server	The replication server from which the routine was run.	Can be NULL.
tag	A character string identifying the grid operations to rerun.	Must be an existing tag. Can be NULL.
target_server	The replication server on which to rerun the routine.	Can be NULL.

Usage

Commands that you run through the grid might fail on one or more servers in the grid. Use the **ifx_grid_redo()** procedure to rerun commands that failed. For example, if you create a fragmented table through the grid and one of the grid servers does not have one of the dbspaces into which the table is fragmented, the command fails on that server. After you add the required dbspace to the server, run the **ifx_grid_redo()** procedure to create the fragmented table on that server.

You can specify from which source server the commands were run, the command ID, the target server on which the commands failed, or the identifying tag for commands that failed.

Use the **force** argument to rerun commands that succeeded.

You must run this routine as an authorized user on an authorized server, as specified by the **cdr grid enable** command.

Example

The following example reruns failed commands on every server in the grid on which those commands failed:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1');
```

The following example reruns the command with the ID of 21 that originated server **cdr1** on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1', 'cdr1', 'cdr4', NULL, '21');
```

The following example reruns all commands that failed on server **cdr4**:

```
EXECUTE PROCEDURE ifx_grid_redo('grid1', NULL, 'cdr4');
```

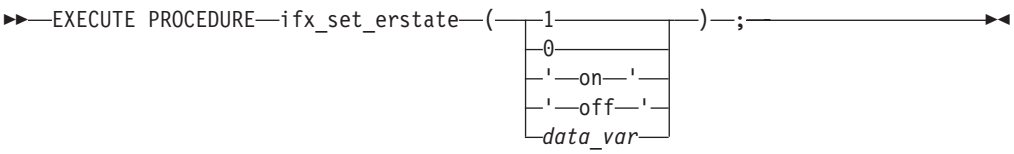
Related tasks:

"Rerunning failed grid routines" on page 7-13

ifx_set_erstate() procedure

The **ifx_set_erstate()** procedure controls whether database operations are replicated.

Syntax



Element	Purpose	Restriction
data_var	Variable holding the value that a function returned	

Usage

Use the **ifx_set_erstate()** procedure to enable or disable replication during a transaction. During normally replicated transactions, use the **ifx_set_erstate()** procedure to enable the recapture of a transaction after it has been replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Replication can only be enabled on tables that are participants in an existing replicate. To enable replication, set the **ifx_set_erstate()** procedure to `1` or `'on'`. To disable replication, set the **ifx_set_erstate()** procedure to `0` or `'off'`. To set replication to a previous state that was saved by the **ifx_get_erstate()** function, set the **ifx_set_erstate()** procedure to the name of the variable returned by the **ifx_get_erstate()** function.

Example

The following example enables replication in the transaction:

```
EXECUTE PROCEDURE ifx_set_erstate(1);
```

The following example resets the replication state to a previous state that was saved by the **ifx_get_erstate()** function in the **curstate** variable:

```
EXECUTE PROCEDURE ifx_set_erstate(curstate);
```

Related concepts:

"Recapture replicated transactions" on page 8-28

Related tasks:

"Enabling replication within a grid transaction" on page 7-14

Related reference:

"ifx_get_erstate() function" on page C-1

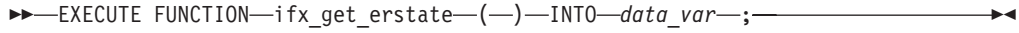
Appendix D. Enterprise Replication routines

Enterprise Replication routines used to control if a replicated transaction is recaptured.

ifx_get_erstate() function

The **ifx_get_erstate()** function indicates whether replication is enabled for the transaction in which it is run.

Syntax



Element	Purpose	Restriction
data_var	Variable to receive the value that the function returns	

Usage

Use the **ifx_get_erstate()** function to obtain the state of replication within a transaction. You can use the state information saved in the variable as input to the **ifx_set_erstate()** procedure.

Return value

A return value of 1 indicates that the current transaction is replicating data.

A return value of 0 indicates that the current transaction is not replicating data.

Example

The following example obtains the replication state and stores it in the **curstate** variable:

```
EXECUTE FUNCTION ifx_get_erstate() INTO curstate;
```

Related concepts:

"Recapture replicated transactions" on page 8-28

Related tasks:

"Enabling replication within a grid transaction" on page 7-14

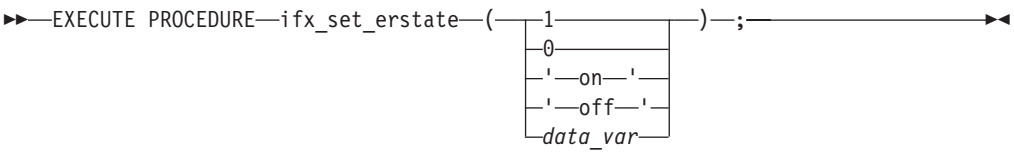
Related reference:

"ifx_set_erstate() procedure" on page C-10

ifx_set_erstate() procedure

The **ifx_set_erstate()** procedure controls whether database operations are replicated.

Syntax



Element	Purpose	Restriction
data_var	Variable holding the value that a function returned	

Usage

Use the **ifx_set_erstate()** procedure to enable or disable replication during a transaction. During normally replicated transactions, use the **ifx_set_erstate()** procedure to enable the recapture of a transaction after it has been replicated. You must reset the replication state back to the default at the end of the transaction or replication loops indefinitely.

Replication can only be enabled on tables that are participants in an existing replicate. To enable replication, set the **ifx_set_erstate()** procedure to `1` or `'on'`. To disable replication, set the **ifx_set_erstate()** procedure to `0` or `'off'`. To set replication to a previous state that was saved by the **ifx_get_erstate()** function, set the **ifx_set_erstate()** procedure to the name of the variable returned by the **ifx_get_erstate()** function.

Example

The following example enables replication in the transaction:

```
EXECUTE PROCEDURE ifx_set_erstate(1);
```

The following example resets the replication state to a previous state that was saved by the **ifx_get_erstate()** function in the **curstate** variable:

```
EXECUTE PROCEDURE ifx_set_erstate(curstate);
```

Related concepts:

"Recapture replicated transactions" on page 8-28

Related tasks:

"Enabling replication within a grid transaction" on page 7-14

Related reference:

"ifx_get_erstate() function" on page C-1

Appendix E. onstat Command Reference

You can monitor and debug Enterprise Replication activity using **onstat** commands.

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 *IBM Informix Administrator's Reference*. For information on SMI tables specific to Enterprise Replication, see Appendix G, "SMI Tables for Enterprise Replication Reference," on page G-1.

Related concepts:

"Monitor Enterprise Replication" on page 9-1

onstat -g ath

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 *IBM 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	preDDR	Runs during queue recovery to monitor the log and sets blockout mode if the log position advances too far before replication resumes.
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 per large transaction	CDRPager	Performs the physical IO for the temporary smart large object that holds paged transaction records. Grouper paging is activated for a transaction when its size is 10 percent of the value of SHMVIRTSIZE or CDR_QUEUEMEM or when it includes more than 100,000 records.
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 (data sync 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.

Number of Threads	Thread Name	Thread Description
0 or 1	CDRM_Monitor	Monitors and adjusts data sync 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

Prints information from the Enterprise Replication global catalog.

Modifier	Description
replname	The name of a replicate

Usage

The global catalog contains a summary of information about the defined servers, replicates, and replicate sets on each of the servers within the domain. If a replicated table is undergoing an alter operation, the **onstat -g cat** command shows that it is in alter mode. For example, use this command to determine:

- How many servers and how many replicates are configured
- Which table matches a given replicate
- Whether a server is a root or leaf server
- The current bitmap mask for a given server. You can use the bitmap mask with the output from the **onstat -g rqm** command to determine which server Enterprise Replication is waiting on for an acknowledgment.

You can set the scope of the output by specifying one of the following options to **onstat -g cat**:

- **full**: (Default) Prints expanded information for both replicate servers and replicates.
- *replname*: Prints information on the specified replicate only.
- **repls**: Prints information on replicates only.
- **servers**: Prints information on servers only.

This sample output from the **onstat -g cat repls** command shows that the table **tab** is in alter mode. The replicate **rep1** is defined on this table, its replicate ID is 6553601.

```
GLOBAL-CATALOG CACHE STATISTICS
REPLICATES
------------------
Parsed statements:
    Id 6553601 table tab
    Id 6553602 table tab12
Inuse databases: test(2)
  Name: rep1, Id: 6553601 State: ACTIVE Flags: 0x800000 ALTERMODE
```

```
        use 0 lastexec Wed Dec 31 18:00:00 1969
    Local Participant: test:nagaraju.tab
    Attributes: TXN scope, Enable ATS, Enable RIS, all columns
        sent in updates
    Conflict resolution: [TIMESTAMP]
    Column Mapping: ON, columns INORDER, offset 8, uncomp_len 12
    Column Name Verifcation: ON
    No Replicated UDT Columns
  Name: rep12, Id: 6553602 State: ACTIVE Flags: 0x800000 use 0
        lastexec Wed Dec 31 18:00:00 1969
    Local Participant: test:nagaraju.tab12
    Attributes: TXN scope, Enable ATS, Enable RIS, all columns
        sent in updates
    Conflict resolution: [TIMESTAMP]
    Column Mapping: ON, columns INORDER, offset 8, uncomp_len 2064
    Column Name Verifcation: ON
    No Replicated UDT Columns
```

The following replicate information shows that the replicate belongs to a grid replicate set. **UTF8** indicates that code set conversion between replicates is enabled.

```
Name: grid_6553604_100_3, Id: 6553605 State: ACTIVE Flags: 0x900000 UTF8 GRID
        use 0 lastexec Wed Dec 31 18:00:00 1969

        Local Participant: tdb:nagaraju.t1
        Attributes: ROW scope, Enable RIS, all columns sent in updates
        Conflict resolution[Prim::Sec]: [ALWAYSAPPLY]
        Column Mapping: OFF
        Column Name Verifcation: ON
        No Replicated UDT Columns
```

This sample output from the **onstat -g cat servers** command shows that the server **g_bombay** and **g_delhi** are active; neither one is a hub or a leaf server, and both have ATS and RIS files generated in XML format.

```
GLOBAL-CATALOG CACHE STATISTICS

SERVERS
-------------------
  Current server  : Id 200, Nm g_bombay
  Last server slot: (0, 2)
  # free slots    : 0
  Broadcast map   : <[0005]>
  Leaf server map : <[0000]>
  Root server map : <[0006]>
  Adjacent server map: <[0004]>
    Id: 200, Nm: g_bombay, Or: 0x0002, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 00, ishub: FALSE, isleaf: FALSE
      subtree map: <empty>
      atsrisformat=xml

    Id: 100, Nm: g_delhi, Or: 0x0004, off: 0, idle: 0, state Active
      root Id: 00, forward Id: 100, ishub: FALSE, isleaf: FALSE
      subtree map: <empty>
      atsrisformat=xml
```

Related tasks:

"Viewing grid information" on page 7-7

Related reference:

"onstat -g cdr"

onstat -g cdr

Prints the output for all of the Enterprise Replication statistics commands.

Usage

The output of the **onstat -g cdr** command is a combination of the following Enterprise Replication **onstat** command outputs:

- **onstat -g cat**
- **onstat -g grp**
- **onstat -g que**
- **onstat -g rqm**
- **onstat -g nif all**
- **onstat -g rcv**
- **onstat -g dss**
- **onstat -g dtc**
- **onstat -g rep**

Related reference:

"onstat -g cat" on page E-2

"onstat -g grp" on page E-8

"onstat -g que" on page E-14

"onstat -g rqm" on page E-17

"onstat -g nif" on page E-13

"onstat -g rcv" on page E-15

"onstat -g dss" on page E-7

"onstat -g dtc" on page E-8

"onstat -g rep" on page E-17

onstat -g cdr config

Prints the settings of Enterprise Replication configuration parameters and environment variables that can be set with the CDR_ENV configuration parameter.

This command has the following formats:

```
onstat -g cdr config
onstat -g cdr config long
onstat -g cdr config parameter_name
onstat -g cdr config parameter_name long
onstat -g cdr config CDR_ENV
onstat -g cdr config CDR_ENV long
onstat -g cdr config CDR_ENV variable_name
onstat -g cdr config CDR_ENV variable_name long
```

The **long** option prints additional information about settings that can be useful for IBM Support.

The following table describes *parameter_name* and *variable_name*.

Modifier	Description
parameter_name	The name of an Enterprise Replication configuration parameter
variable_name	The name of an Enterprise Replication environment variable

If you use **onstat -g cdr config** without any options, the settings of all Enterprise Replication configuration parameters and environment variables are included in the output. If you specify the CDR_ENV configuration parameter without an environment variable name, all Enterprise Replication environment variables are included in the output.

The following sample output of the **onstat -g cdr config ENCRYPT_CDR** command shows the setting of the ENCRYPT_CDR configuration parameter:

```
onstat -g cdr config ENCRYPT_CDR


ENCRYPT_CDR configuration setting:          0
```

The following sample output of the **onstat -g cdr config CDR_ENV** command shows the settings of all Enterprise Replication environment variables:

```
onstat -g cdr config CDR_ENV

CDR_ENV environment variable settings:
    CDR_LOGDELTA:
        CDR_LOGDELTA configuration setting:           0
    CDR_PERFLOG:
        CDR_PERFLOG configuration setting:            0
    CDR_ROUTER:
        CDR_ROUTER configuration setting:             0
    CDR_RMSCALEFACT:
        CDR_RMSCALEFACT configuration setting:        0
    CDRSITES_731:
        CDRSITES_731 configuration setting:             [None configured]
    CDRSITES_92X:
        CDRSITES_92X configuration setting:             [None configured]
    CDRSITES_10X:
        CDRSITES_10X configuration setting:             [None configured]
```

The following sample output of the **onstat -g cdr config** command shows the settings of all Enterprise Replication configuration parameters and CDR_ENV environment variables:

```
onstat -g cdr config

CDR_DBSPACE:
    CDR_DBSPACE configuration setting:               rootdbs
CDR_DSLOCKWAIT:
    CDR_DSLOCKWAIT configuration setting:            5
CDR_EVALTHREADS:
    CDR_EVALTHREADS configuration setting:           1, 2
CDR_MAX_DYNAMIC_LOGS:
    CDR_MAX_DYNAMIC_LOGS configuration setting:      0
CDR_NIFCOMPRESS:
    CDR_NIFCOMPRESS configuration setting:           0
CDR_QDATA_SBSPACE:
    CDR_QDATA_SBSPACE configuration setting:         cdrsbsp
CDR_QHDR_DBSPACE:
    CDR_QHDR_DBSPACE configuration setting:          rootdbs
CDR_QUEUEMEM:
    CDR_QUEUEMEM configuration setting:              4096
CDR_SERIAL:
    CDR_SERIAL configuration setting:                0, 0
CDR_SUPPRESS_ATSRISWARN:
```

```
    CDR_SUPPRESS_ATSRISWARN configuration setting:    [None suppressed]
ENCRYPT_CDR:
    ENCRYPT_CDR configuration setting:               0
ENCRYPT_CIPHERS:
    ENCRYPT_CIPHERS configuration setting:           [None configured]
ENCRYPT_MAC:
    ENCRYPT_MAC configuration setting:               [None configured]
ENCRYPT_MACFILE:
    ENCRYPT_MACFILE configuration setting:           [None configured]
ENCRYPT_SWITCH:
    ENCRYPT_SWITCH configuration setting:            0,0
CDR_ENV environment variable settings:
    CDR_LOGDELTA:
        CDR_LOGDELTA configuration setting:          0
    CDR_PERFLOG:
        CDR_PERFLOG configuration setting:           0
    CDR_ROUTER:
        CDR_ROUTER configuration setting:            0
    CDR_RMSCALEFACT:
        CDR_RMSCALEFACT configuration setting:       0
    CDRSITES_731:
        CDRSITES_731 configuration setting:            [None configured]
    CDRSITES_92X:
        CDRSITES_92X configuration setting:            [None configured]
    CDRSITES_10X:
        CDRSITES_10X configuration setting:            [None configured]
```

Related tasks:

"Dynamically Modifying Configuration Parameters for a Replication Server" on page 8-1

onstat -g ddr

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.

You can use the information from the **onstat -g ddr** command to monitor *replay position* in the log file and ensure replay position is never overwritten (which can cause loss of data). The replay position is the point from where, if a system failure occurs, Enterprise Replication starts re-reading the log information into the log update buffers. All the transactions generated before this position at all the target servers have been applied by Enterprise Replication or safely stored in stable queue space. As messages are acknowledged or stored in the stable queue, the replay position should advance. If you notice that replay position is not advancing, this can mean that the stable queue is full or a remote server is down.

The **onstat -g ddr** output shows you a snapshot of the replay position, the *snoopy position*, and the *current position*. The snoopy position identifies the position of the **ddr_snoopy** thread in the logical logs. The **ddr_snoopy** has read the log records up until this point. The current position is the position where the server has written its last logical log record.

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 (**ddr_snoopy**) position, which means that data will not be replicated. If this occurs, you must manually resynchronize the source and target databases.

To avoid these problems, follow these guidelines:

- Have 24 hours of online log space available.
- Keep the log file size consistent. Instead of having a single large log file, implement several smaller ones.

- Avoid switching logical logs more than once per hour.
- Keep some distance between LTXHWM (long-transaction high-watermark) and LTXEHWM (long-transaction, exclusive-access, high-watermark).

You can configure one or more actions to occur if the current position reaches the log needs position by setting the CDR_LOG_LAG_ACTION configuration parameter.

The following sample output from the **onstat ddr** command shows the replay position, snoopy position, and current position highlighted.

```
DDR -- Running --

# Event  Snoopy   Snoopy   Replay   Replay   Current  Current
Buffers   ID      Position  ID      Position   ID     Position
2064     35       2ae050   34       121018   55       290000

Log Pages Snooped:
      From      From      From Staging     Tossed
     Cache      Disk      File             (LBC full)
         0         0       19704          0

CDR log records ignored   : 0
DDR log lag state   : On
Current DDR log lag action   : logstage
DDR log staging disk space usage  :0.26%
Maximum disk space allowed for log staging :1048576 KB
Maximum disk space ever used for log staging :2746.98 KB
Current staged log file count :21
Total dynamic log requests: 0

DDR events queue

Type   TX id    Partnum  Row id
```

Related reference:

"cdr view" on page A-169

onstat -g dss

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.

In the following example, only one data sync thread is currently processing the replicated data. It has applied a total of one replicated transaction and the transaction was applied at 2004/09/13 18:13:10. The Processed Time field shows the time when the last transaction was processed by this data sync thread.

```
-- Up 00:00:28 -- 28672 Kbytes
DS thread statistic
cmtTime    Tx       Tx        Tx       Last Tx
Name       < local  Committed Aborted  Processed  Processed Time
---------- -------  --------- -------  ---------  -----------------
CDRD_1     0        1         0        1          (1095117190) 2004/09/13
18:13:10
            Tables (0.0%):
            Databases: test
CDR_DSLOCKWAIT = 1
CDR_DSCLOSEINTERVAL = 60
```

Related reference:

"onstat -g cdr" on page E-4

onstat -g dtc

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 IBM Software Support.

In the following example, the thread name of the delete table cleaner is **CDRDTCleaner**. The total number of rows deleted is **1**. The last activity on this thread occurred at 2010/08/13 18:47:19. The delete table for replicate **rep1** was last cleaned at 2010/08/13 18:28:25.

```
-- Up 00:59:15 -- 28672 Kbytes
-- Delete Table Cleanup Status as of (1095119368) 2010/08/13 18:49:28
thread           = 49 <CDRDTCleaner>
      rows deleted     = 1
      lock timeouts    = 0
      cleanup interval = 300
      list size        = 3
      last activity    = (1095119239) 2010/08/13 18:47:19

Id      Database                          Last Cleanup Time
        Replicate            Server              Last Log Change
=========================================================
000001  test                              (1095118105) 2010/09/13
  18:28:25
        rep1                 g_bombay            (1095118105) 2010
        /08/13 18:28:25
        rep1                 g_delhi             (1095118105) 2010
        /08/13 18:28:25
000002  test                              <never cleaned>
```

Related reference:

"onstat -g cdr" on page E-4

onstat -g grp

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 IBM Software 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
pager	Prints grouper paging statistics
R	Prints grouper replicate statistics
S	Prints grouper serial list head (The serial list head is the first transaction in the list, that is, the next transaction that will be placed in the send queue.)
Sl	Prints grouper serial list (The serial list is the list of transactions, in chronological order.)
Sx	Prints grouper serial list, expands open transactions
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.

The rest of this section contains sample output from various **onstat -g grp** *modifier* commands. The following sample shows output for the **onstat -g grp** command.

```
Grouper at 0xb014018:
Last Idle Time: (1095122236) 2010/09/13 19:37:16
RSAM interface ring buffer size: 528
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 48
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 5
Log update buffer memory in use: 0
Max log update buffer memory used at once: 320
Updates from Log: 16
Log update links allocated: 512
Blob links allocated: 0
```

```
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
Last Tx to Queuer began : (1095118105) 2010/09/13 18:28:25
Last Tx to Queuer ended : (1095118105) 2010/09/13 18:28:25
Last Tx to Queuer log ID, position: 12,23
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 2
Tx returned from Queuer: 2
Events sent to Queuer: 7
Events returned from Queuer: 7
Total rows sent to Queuer: 2
Open Tx array size: 1024
Table 'tab' at 0xae8ebb0 [ CDRShadow ]
Table 'tab12' at 0xae445e0 [ CDRShadow ]

Grouper Table Partitions:
  Slot  312...
    'tab' 1048888
  Slot  770...
    'tab12' 3145730
  Slot 1026...
    'tab12' 4194306
Repl links on global free list: 2
Evaluators: 3
  Evaluator at 0xb03d030 ID 0 [Idle:Idle] Protection:unused
    Eval iteration: 1264
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xb03d080
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xb03d0d8 ID 1 [Idle:Idle] Protection:unused
    Eval iteration: 1265
    Updates evaluated: 2
    Repl links on local free list: 254
    UDR environment table at 0xb03d128
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xb03d180 ID 2 [Idle:Idle] Protection:unused
    Eval iteration: 1266
    Updates evaluated: 4
    Repl links on local free list: 256
    UDR environment table at 0xb03d1d0
        Number of environments:          0
        Table memory limit    :      25165
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
        Total Free Repl links 768

Replication Group 6553601 at 0xb0a8360
  Replication at 0xb0a82b0 6553601:6553601 (tab) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 10
Replication Group 6553602 at 0xb0a8480
  Replication at 0xb0a83d0 6553602:6553602 (tab12)[Ignore Stopped NotifyDS FullRowOn]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 16
```

The following example shows output for the **onstat -g grp E** command. The field **Evaluators: 4** indicates that there are four evaluation threads configured for the system.

```
Repl links on global free list: 0 Evaluators: 4
  Evaluator at 0xba71840 ID 0 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba71890
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0
  Evaluator at 0xba718f0 ID 1 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba71940
        Number of environments:           0
        Table memory limit    :       16777
        Table memory used     :           0
        SAPI memory limit     :      131072
        SAPI memory used      :           0
        Count failed UDR calls:           0

  Evaluator at 0xba8c260 ID 2 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xba8c2b0
        Number of environments:           0
        Table memory limit    :      16777
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
  Evaluator at 0xbaac2a0 ID 3 [Idle:Idle] Protection: unused
    Eval iteration: 1007
    Updates evaluated: 0
    Repl links on local free list: 256
    UDR environment table at 0xbaac2f0
        Number of environments:           0
        Table memory limit    :      16777
        Table memory used     :          0
        SAPI memory limit     :     131072
        SAPI memory used      :          0
        Count failed UDR calls:          0
Total Free Repl links 1024
```

The following example shows output for the **onstat -g grp G** command.

```
Grouper at 0xb8ab020:
Last Idle Time: (1095115397) 2010/09/13 17:43:17
RSAM interface ring buffer size: 1040
RSAM interface ring buffer pending entries: 0
Eval thread interface ring buffer size: 64
Eval thread interface ring buffer pending entries: 0
Log update buffers in use: 0
Max log update buffers used at once: 1
Log update buffer memory in use: 0
Max log update buffer memory used at once: 64
Updates from Log: 1
Log update links allocated: 512
Blob links allocated: 0
Conflict Resolution Blocks Allocated: 0
Memory pool cache: Empty
```

The following example shows output for the **onstat -g grp P** command. In the following example, the grouper is evaluating rows for the **account**, **teller** and **customer** tables.

```
Table 'teller' at 0xb851480 [ CDRShadow VarChars ]
Table 'account' at 0xb7faad8 [CDRShadow VarChars VarUDTs Floats
       Blobs]
Table 'customer' at 0xbbe67a8 [CDRShadow VarChars VarUDTs]
Grouper Table Partitions:
  Slot  387...
    'account' 1048707
  Slot  389...
    'teller' 1048709
  Slot  394...
    'customer' 1048714
```

The following example shows output for the **onstat -g grp pager** command. The sample output shows the grouper large transaction evaluation statistics.

```
Grouper Pager statistics:
Number of active big transactions: 0
Total number of big transactions processed: 0
Spool size of the biggest transaction processed: 0 Bytes
```

The following example shows output for the **onstat -g grp R** command. In this example, the grouper is configured to evaluate rows for replicates with IDs **6553601** and **6553602** (you can use the **onstat -g cat repls** command to obtain the replicate names). The **Ignore** attribute of replicate ID **6553602** shows that the grouper is currently not evaluating rows for this replicate. This can happen if the replicate state is not ACTIVE. You can obtain the replicate state using the **onstat -g cat repls** command.

```
Replication Group 6553601 at 0xb0a8360
  Replication at 0xb0a82b0 6553601:6553601 (tab) [ NotifyDS FullRowOn ]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 10
Replication Group 6553602 at 0xb0a8480
  Replication at 0xb0a83d0 6553602:6553602 (tab12)[Ignore Stopped NotifyDS FullRowOn]
    Column Information [ CDRShadow VarUDTs InOrder Same ]
      CDR Shadow: offset 0, size 8
      In Order: offset 8, size 16
```

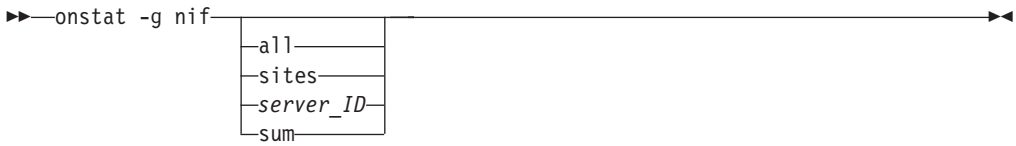
The following example shows output for the **onstat -g grp T** command. In this example, the grouper evaluated and queued 1 transaction to the send queue. The **Tx sent to Queuer** field shows the total number of transactions evaluated and queued to the send queue for propagating to all the replicate participants. The **Total rows sent to Queuer** field shows the total number of rows queued to the send queue for propagating to all the replicate participants.

```
Last Tx to Queuer began : (1095116676) 2010/09/13 18:04:36
Last Tx to Queuer ended : (1095116676) 2010/09/13 18:04:36
Last Tx to Queuer log ID, position: 5,3236032
Open   Tx: 0
Serial Tx: 0
Tx not sent: 0
Tx sent to Queuer: 1
Tx returned from Queuer: 0
Events sent to Queuer: 0
Events returned from Queuer: 0
Total rows sent to Queuer: 1
Open Tx array size: 1024
```

Related reference:
"onstat -g cdr" on page E-4

onstat -g nif

Prints statistics about the network interface.



The output shows which sites are connected and provides a summary of the number of bytes sent and received by each site. This can help you determine if a site is not sending or receiving bytes.

The **-g nif** option is used primarily as a debugging tool and by Technical Support.

The following table describes the options for **onstat -g nif**:

Option	Action
all	Prints the sum and the sites.
sites	Prints the NIF site context blocks.
server_ID	Prints information about the replication server with that server ID.
sum	Prints the sum of the number of buffers sent and received for each site.

Example Output

The following example shows output for the **onstat -g nif** command. In this example, the local server is connected to the server group **g_bombay** and its CDR ID is **200**. The connection status is running. The connection between the two servers is running, but the replication state on the **g_bombay** server is suspended. The server group **g_bombay** internal NIF version is **9**. The local server has sent three messages to the server **g_bombay** and it has received two messages from **g_bombay**.

```
$ onstat -g nif

NIF anchor Block: af01610
       nifGState         RUN
          RetryTimeout   300

CDR connections:
 Id    Name       State    Version    Sent   Received
 ----------------------------------------------------
 200 g_bombay     RUN,SUSPEND    9       3          2
```

Output Description

NIF anchor Block
The address of the network storage block.

nifGState
The connection state.

RetryTimeout
The number of seconds before Enterprise Replication attempts to retry a dropped connection.

Id The Enterprise Replication ID number for the server.

Name The name of the server group.

State The connection state between the local server and the listed server. If multiple states are shown the second state designates the replication state.

Version
The internal version number of the NIF component on the listed server.

Sent The number of messages the local server has sent to the listed server.

Received
The number of messages received by the local server from the listed server.

Related reference:

"onstat -g cdr" on page E-4

onstat -g que

Prints statistics that are common to all queues.

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 Technical Support.

In the following example, **Element high water mark** shows the maximum size of the transaction buffer header data (metadata) allowed in memory, shown in kilobytes. **Data high water mark** shows the maximum size of transactions for user data allowed in memory, shown in kilobytes.

```
CDR Queuer Statistics:
  Queuer state            : 2
  Local server            : 100
  Element high water mark : 131072
  Data high water mark    : 131072
  # of times txns split   : 0
  Total # of split txns   : 0
  allowed log delta       : 30
  maximum delta detected  : 4
  Control Key             : 0/00000007
  Synchronization Key     : 0/00000003
Replay Table:
  Replay Posn (Disk value): 12/00000018 (12/00000018)
  Replay save interval    : 10
  Replay updates          : 10
  Replay # saves          : 17
  Replay last save time   : (1095118157) 2010/09/13 18:29:17
Send Handles
  Server ID               : 200
  Send state,count        : 0,0
  RQM hdl for trg_send: Traverse handle (0xaf8e018) for thread CDRACK_0 at Head_of_Q,
      Flags: None
  RQM hdl for control_send:  Traverse handle (0xaf74018)
    for thread CDRACK_0 at Head_of_Q,  Flags: None
  RQM hdl for sync_send: Traverse handle (0xadc6018) for thread CDRACK_0 at Head_of_Q,
      Flags: None
  Server ID               : 200
  Send state,count        : 0,0
  RQM hdl for trg_send: Traverse handle (0xac8b018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  RQM hdl for control_send: Traverse handle (0xb1ce018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
  RQM hdl for sync_send: Traverse handle (0xadc5018) for thread CDRACK_1 at Head_of_Q,
      Flags: None
```

```
Server ID              : 200
Send state,count       : 0,0
RQM hdl for trg_send: Traverse handle (0xaea71d8) for thread CDRNsA200 at Head_of_Q,
    Flags: None
RQM hdl for ack_send: Traverse handle (0xae8c1d8) for thread CDRNsA200 at Head_of_Q,
    Flags: None
RQM hdl for control_send: Traverse handle (0xae9e1d8) for thread CDRNsA200 at Head_of_Q,
    Flags: None
```

Related reference:

"onstat -g cdr" on page E-4

onstat -g rcv

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 Manager global section. In this section, the following fields have the meanings shown:

Field	Description
cdrRM_DSParallelPL	Shows the current level of Apply Parallelism, 0 (zero) being the highest
cdrRM_DSNumLockTimeout cdrRM_DSNumLockRB cdrRM_DSNumDeadLocks	Indicate the number of collisions between various apply threads
cdrRM_acksinList	Shows acknowledgments that have been received but not yet processed

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
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

The Statistics by Source section of the **onstat -g rcv** command shows the following information for each source server. For each replicate ID:

- The number of transactions applied from the source servers
- The number of inserts, deletes, and updates within the applied transactions
- The timestamp of the most recently applied transaction on the target server
- The timestamp of the commit on the source server for the most recently applied transaction

The **-g rcv** option is used primarily as a debugging tool and by Technical Support. If you suspect that acknowledgment messages are not being applied, you can use this option to check.

The following example shows output for the **onstat -g rcv full** command.

```
Receive Manager global  block 0D452018
    cdrRM_inst_ct:                    5
    cdrRM_State:              00000000
    cdrRM_numSleepers:        3
    cdrRM_DsCreated:          3
    cdrRM_MinDSThreads:       1
    cdrRM_MaxDSThreads:       4
    cdrRM_DSBlock             0
    cdrRM_DSParallelPL        0
    cdrRM_DSFailRate          0.000000
    cdrRM_DSNumRun:           35
    cdrRM_DSNumLockTimeout    0
    cdrRM_DSNumLockRB         0
    cdrRM_DSNumDeadLocks      0
    cdrRM_DSNumPCommits       0
    cdrRM_ACKwaiting          0
    cdrRM_totSleep:           77
    cdrRM_Sleeptime:          153
    cdrRM_Workload:           0
    cdrRM_optscale:           4
    cdrRM_MinFloatThreads:    2
    cdrRM_MaxFloatThreads:    7
    cdrRM_AckThreadCount:     2
    cdrRM_AckWaiters:         2
    cdrRM_AckCreateStamp:Wed Sep 08 11:47:49 2010
    cdrRM_DSCreateStamp: Wed Sep 08 14:16:35 2010
    cdrRM_acksInList:         0
    cdrRM_BlobErrorBufs:      0

Receive Parallelism Statistics
Srvr Tot.Txn. Pnding Active MaxPnd MaxAct AvgPnd AvgAct CommitRt
   1     35        0      0     21      3   7.00   1.63    0.00
   5      3        0      0      1      1   1.00   1.00    0.02
   6      6        0      0      1      1   1.00   1.00    0.21
Tot Pending:0 Tot Active:0 Avg Pending:5.77 Avg Active:1.50
Commit Rate:0.01

Time Spent In RM Parallel Pipeline Levels
Lev. TimeInSec  Pcnt.
   0     17405 100.00%
   1         0   0.00%
   2         0   0.00%

Statistics by Source
Server 1
Repl  Txn Ins Del Upd Last Target Apply   Last Source Commit
65541  23   0   1 616 2010/09/08 14:20:15 2010/09/08 14:20:15
65542  11   0   0 253 2010/09/08 14:19:33 2010/09/08 14:19:33
65545   1   0  67   0 2010/09/08 14:20:37 2010/09/08 14:20:37
Server 5
```

```
Repl  Txn Ins Del Upd Last Target Apply   Last Source Commit
65541   3   0   0  81 2010/09/08 16:36:10 2010/09/08 16:36:09
Server 6
Repl  Txn Ins Del Upd Last Target Apply   Last Source Commit
65548   6   0   0  42 2010/09/08 16:37:59 2010/09/08 16:37:58
```

Related reference:

"onstat -g cdr" on page E-4

onstat -g rep

Prints events that are in the queue for the schedule manager.

The **-g rep** option is used primarily as a debugging tool and by 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*.

The following example shows sample output for the **onstat -g rep** command:

```
Schedule manager Cb: add7e18 State: 0x8100 <CDRINIT,CDRRUNNING>

Event        Thread             When
=========================================
CDRDS          CDREvent       00:00:20
```

Related reference:

"cdr swap shadow" on page A-159

"onstat -g cdr" on page E-4

onstat -g rqm

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. You can choose to print information for all queues or for just one queue by using one of the modifiers described below.

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
SBSPACES	Prints detailed statistical information about the sbspaces configured for CDR_QDATA_SBSPACE.
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. When you select the **SBSPACES** modifier, the command prints information about the sbspaces being used for replication, including how full those sbspaces are.

The other modifiers of the **onstat -g rqm** command are used primarily as a debugging tool and by Technical Support.

The output for the SENDQ modifier contains the following sections:

- The current statistics section (`Transaction spool name` through `Pending Txn Data`): Contains information about the current contents of the queue, such as how many bytes are contained in the queue, how many transactions are in the queue, how many transactions are currently in memory, how many have been spooled to disk, how many exist only on disk, and so on. The `Insert Stamp` field value is used to maintain the order of the transactions within the queue. The `Size of Data in queue` field shows the size of the queue when combining the in-memory transactions with the spool-only transactions. The `Pending Txn Buffers` field contains information about transactions that are in the process of being queued into the send queue.
- The historical statistics section (`Max Real memory data used` through `Total Txn Lookups`): contains a summary of what has been placed in the queue in the past. The `Max Real memory data used` field contains the largest in memory size of the queue. The `Total Txn Recovered` field shows the transactions that existed only in the spool when the server was started. The `Total Txns deleted` field shows the number of transactions that have been removed from the queue. The `Total Txns duplicated` field contains the number of times attempted to queue a transaction that had already been processed. The `Total Txn Lookups` field is a counter of the number of times that an Enterprise Replication thread attempted to read a transaction.
- The `Progress Table` section: contains information on what is currently queued, to which server it is queued for, and what has been acknowledged from each of the participants of the replicate. The first part of the progress table section is a summary. Below the summary section is a list of the servers and group entries that contain what is currently queued for each server, what has been sent to the remote server, and what has been acknowledged from the remote server. The contents of the `ACKed` and `Sent` columns contains the key of the last transaction that was acknowledged from the remote server or sent to that server. The key is

a multi-part number consisting of *source_node/unique_log_id/logpos/incremental number*. The transaction section contains the first and last transaction in the queue that are currently in memory. The `NeedAck` field shows from which server the transaction is waiting for an acknowledgment. You can use this bitmap mask with the output from the **onstat -g cat** command to determine the name of the server which server Enterprise Replication is waiting on for an acknowledgment.

- The `Transverse handle` section: contains the position within the queue that any thread is currently processing. Each thread that attempts to read a transaction from the queue, or to place a transaction into the queue must first allocate a handle. This handle is used to maintain the positioning within the queue.

The following example shows output for the **onstat -g rqm SENDQ** command.

```
> onstat  -g rqm SENDQ


CDR Reliable Queue Manager (RQM) Statistics:

RQM Statistics for Queue (0xb956020) trg_send
 Transaction Spool Name: trg_send_stxn
 Insert Stamp: 9/0
 Flags: SEND_Q, SPOOLED, PROGRESS_TABLE, NEED_ACK
 Txns in queue:             0
 Log Events in queue:       0
 Txns in memory:            0
 Txns in spool only:        0
 Txns spooled:              0
 Unspooled bytes:           0
 Size of Data in queue:     0 Bytes
 Real memory in use:        0 Bytes
 Pending Txn Buffers:       0
 Pending Txn Data:          0 Bytes
 Max Real memory data used: 385830 (4194304) Bytes
 Max Real memory hdrs used  23324 (4194304) Bytes
 Total data queued:         531416 Bytes
 Total Txns queued:         9
 Total Txns spooled:        0
 Total Txns restored:       0
 Total Txns recovered:      0
 Spool Rows read:           0
 Total Txns deleted:        9
 Total Txns duplicated:     0
 Total Txn Lookups:         54

 Progress Table:
        Progress Table is Stable
                On-disk table name.............:         spttrg_send
                Flush interval (time)..........:         30
                Time of last flush.............:         1207866706
                Flush interval (serial number):          1000
                Serial number of last flush...:          1
                Current serial number..........:         5

Server    Group Bytes Queued      Acked                      Sent
------------------------------------------------------------------------------
    20  0xa0002      12 efffffff/efffffff/efffffff/efffffff -  a/e/1510a1/0
    20  0xa0003                  0 a/e/4ca1b8/0             -  a/e/4ca1b8/0
    30  0xa0004                  0 a/e/4ca1b8/0             -  a/e/4ca1b8/0
    20  0xa0004                  0 a/e/4ca1b8/0             -  a/e/4ca1b8/0
    20  0xa0001                  0 a/d/6e81f8/0             -  a/d/6e81f8/0


First Txn (0x0D60C018) Key:  1/9/0x000d4bb0/0x00000000
 Txn Stamp: 1/0, Reference Count: 0.
 Txn Flags: Notify
```

```
Txn Commit Time: (1094670993) 2004/09/08 14:16:33
Txn Size in Queue: 5908
First Buf's (0x0D31C9E8) Queue Flags: Resident
First Buf's Buffer Flags: TRG, Stream
NeedAck: Waiting for Acks from <[0004]>
No open handles on txn.

Last Txn (0x0D93A098) Key:  1/9/0x00138ad8/0x00000000
Txn Stamp: 35/0, Reference Count: 0.
Txn Flags: Notify
Txn Commit Time: (1094671237) 2004/09/08 14:20:37
Txn Size in Queue: 6298
First Buf's (0x0D92FFA0) Queue Flags: Resident
First Buf's Buffer Flags: TRG, Stream
NeedAck: Waiting for Acks from <[0004]>
Traverse handle (0xbca1a18) for thread CDRGeval0 at Head_of_Q,  Flags: None
Traverse handle (0xb867020) for thread CDRACK_1 at Head_of_Q,  Flags: None
Traverse handle (0xbcbd020) for thread CDRGeval1 at Head_of_Q,  Flags: None
Traverse handle (0xbd08020) for thread CDRGeval3 at Head_of_Q,  Flags: None
Traverse handle (0xbe511c8) for thread CDRGeval2 at Head_of_Q,  Flags: None
Traverse handle (0xbe58158) for thread CDRACK_0 at Head_of_Q,  Flags: None
```

The following output is an example of the **onstat -g rqm SBSPACES** command.

```
onstat -g rqm sbspaces

Blocked:DDR


RQM Space Statistics for CDR_QDATA_SBSPACE:
------------------------------------------
name/addr      number    used      free   total    %full   pathname
0x46581c58     5         311       1      312      100     /tmp/amsterdam_sbsp_base
amsterdam_sbsp_base5     311       1      312      100

0x46e54528     6         295       17     312      95      /tmp/amsterdam_sbsp_2
amsterdam_sbsp_26        295       17     312      95

0x46e54cf8     7         310       2      312      99      /tmp/amsterdam_sbsp_3
amsterdam_sbsp_37        310       2      312      99

0x47bceca8     8         312       0      312      100     /tmp/amsterdam_sbsp_4
amsterdam_sbsp_48        312       0      312      100
```

In this example, the sbspaces are all either full or nearly full.

Related tasks:

"Monitoring Disk Usage for Send and Receive Queue Spool" on page 9-16

Related reference:

"onstat -g cdr" on page E-4

onstat -g sync

Prints statistics about the active synchronization process.

The following example shows output for the **onstat -g sync** command.

```
Prim   Sync   St.  Shadow Flag Stat Block  EndBlk
Repl   Source      Repl             Num    Num

655361 20     0    1310729 2    0    592    600
```

Output Description

Prim Repl
Replicate number of the replicate being synchronized

Sync Source
Source server of the sync

St Sync replicate state

Shadow Repl
The shadow replicate used to perform the sync

Flag Internal flags:

- 0x02 = external sync
- 0x04 = shutdown request has been issued
- 0x08 = abort has occurred
- 0x010 = a replicate stop has been requested
- 0X020 = shadow or primary replicate has been deleted

Stat Resync job state

Block num
Last block applied on targets (on source always 0)

EndBlock Num
Last block in resync process. Marks the end of the sync scan on the target. A value of' -2 indicates that the scan is still in progress, and the highest block number is not yet known.

Additional fields for forwarded rows:

ServID Server where forwarded row originated

fwdLog ID
Originator's log ID of the forwarded row

fwdLog POS
Originator's log position of the forwarded row

endLog ID
Operation switches back to normal at this point

endLog POS
Operation switches back to normal at this log position

complete flag
Set to 1 after normal processing resumes for the originating source

onstat -k

Prints information about active locks.

The following example shows output from the **onstat -k** command:

```
Locks
address  wtlist   owner    lklist   type     tblsnum  rowid    key#/bsiz
a095f78  0        a4d9e68  0        HDR+S    100002   203         0
```

In the following output, the number 2 in the last row shows an Enterprise Replication pseudo lock:

```
Locks
address  wtlist    owner    lklist   type    tblsnum  rowid    key#/bsiz
a197ff8  0         5c2db4a8 0           S    100002   204             0
a198050  0         5c2db4a8 a197ff8     S    100002   205             0
a198260  0         5c2f2248 0           S    100002   204             0
a198470  0         5c2e6b78 a198520     S    100002   205             0
a198520  0         5c2e6b78 0           S    100002   204             0
a1986d8  0         5c2ec6e0 a198ba8     S    100002   205             0
a198ba8  0         5c2ec6e0 0           S    100002   204             0
a1993e8  0         5c2f03d0 a19be30     S         2   1c05a           0
```

You can interpret output from this option as follows:

address Is the address of the lock in the lock table

If a user thread is waiting for this lock, the address of the lock appears in the **wait** field of the **onstat -u** (users) output.

wtlist Is the first entry in the list of user threads that is waiting for the lock, if there is one

owner Is the shared-memory address of the thread that is holding the lock

This address corresponds to the address in the **address** field of **onstat -u** (users) output.

lklist Is the next lock in a linked list of locks held by the owner just listed

type Uses the following codes to indicate the type of lock:

HDR Header

B Bytes

S Shared

X Exclusive

I Intent

U Update

IX Intent-exclusive

IS Intent-shared

SIX Shared, intent-exclusive

tblsnum

Is the tblspace number of the locked resource. If the number is less than 10000, it indicates Enterprise Replication pseudo locks.

rowid Is the row identification number

The rowid provides the following lock information:

- If the rowid equals zero, the lock is a table lock.
- If the rowid ends in two zeros, the lock is a page lock.
- If the rowid is six digits or fewer and does not end in zero, the lock is probably a row lock.
- If the rowid is more than six digits, the lock is probably an index key-value lock.

key#/bsiz

Is the index key number, or the number of bytes locked for a VARCHAR lock

If this field contains 'K-' followed by a value, it is a key lock. The value identifies which index is being locked. For example, K-1 indicates a lock on the first index defined for the table.

The maximum number of locks available is specified as LOCKS in the **ONCONFIG** file.

Appendix F. syscdr Tables

These tables in the **syscdr** database contain progress information about consistency checking and synchronization operations.

The replcheck_stat Table

The **replcheck_stat** table contains the progress information for consistency check and synchronization operations that specified a progress report task name.

Column	Type	Purpose
replcheck_id	serial	Unique key for the task and replicate combination.
replcheck_name	varchar(32)	The task name.
replcheck_replname	varchar(128)	The replicate name.
replcheck_type	char(1)	The task type: • C = consistency check • S = synchronization
replcheck_numrows	integer	The total number of rows in the table.
replcheck_rows_processed	integer	The number of rows processed to correct inconsistent rows.
replcheck_status	char(1)	The status of this task: • D = Defined • R = Running • C = Completed • F = Completed, but inconsistent • W = Pending complete
replcheck_start_time	datetime year to second	The time that the sync or check task for the replicate started running.
replcheck_end_time	datetime year to second	The time that sync or check task for the replicate completed.

Related reference:

"cdr stats check" on page A-143

"cdr stats sync" on page A-147

The replcheck_stat_node Table

The **replcheck_stat_node** table contains the progress information for the consistency check and synchronization operations with progress report task names on a particular replication server.

Column	Type	Purpose
replnode_replcheck_id	integer	Server group ID (CDR ID).
replcheck_node_id	integer	Unique key for the task, replicate, and server combination.
replcheck_order	integer	A number to provide consistent ordering for display purposes.
replcheck_node_name	varchar(128)	The name of the replication server.
replnode_table_owner	varchar(128)	The owner of table being synchronized or checked.
replnode_table_name	varchar(128)	The name of the table being synchronized or checked.
replnode_row_count	integer	The number of rows in the participant.
replnode_processed_rows	integer	The number of rows processed to correct inconsistent rows.
replnode_missing_rows	integer	The number of rows on the reference server that do not exist on the target server.
replnode_extra_rows	integer	The number of rows on the target server that do not exist on the reference server.
replnode_mismatched_rows	integer	The number of rows on the target server that are not consistent with the corresponding rows on the reference server.
replnode_extra_child_rows	integer	The number of child rows that required processing on the target nodes.

Related reference:

"cdr stats check" on page A-143

"cdr stats sync" on page A-147

Appendix G. SMI Tables for Enterprise Replication Reference

The system-monitoring interface (SMI) tables in the **sysmaster** database provide information about the state of the database server. Enterprise Replication uses the following SMI tables.

Related concepts:

"Monitor Enterprise Replication" on page 9-1

The syscdr_ats Table

The **syscdr_ats** table contains the first ten lines of the transaction header for each ATS file.

Column	Type	Description
ats_ris	integer	Pseudo row ID.
ats_file	char(128)	ATS file name.
ats_sourceid	integer	CDRID of source server.
ats_source	char(128)	Source server name.
ats_committime	char(20)	Time when the transaction was committed on the source server.
ats_targetid	integer	CDRID of the target server.
ats_target	char(128)	Target server name.
ats_receivetime	char(20)	Time when the transaction was received on the target server .
ats_risfile	char(128)	Corresponding RIS file name.
ats_line1	char(200)	The first line of the transaction header information.
ats_line2	char(200)	The second line of the transaction header information.
ats_line3	char(200)	The third line of the transaction header information.
ats_line4	char(200)	The fourth line of the transaction header information.
ats_line5	char(200)	The fifth line of the transaction header information.
ats_line6	char(200)	The sixth line of the transaction header information.
ats_line7	char(200)	The seventh line of the transaction header information.
ats_line8	char(200)	The eighth line of the transaction header information.
ats_line9	char(200)	The ninth line of the transaction header information.
ats_line10	char(200)	The tenth line of the transaction header information.

The syscdr_atsdir Table

The **syscdr_atsdir** table contains information about the contents of the ATS directory.

Column	Type	Description
atsd_rid	integer	Pseudo row ID
atsd_file	char(128)	ATS file name
atsd_mode	integer	File mode
atsd_size	integer	File size in bytes

Column	Type	Description
atsd_atime	datetime	Last access time
atsd_mtime	datetime	Last modified time
atsd_ctime	datetime	Create time

The syscdr_ddr Table

The **syscdr_ddr** table contains information about the status of log capture and the proximity or status of transaction blocking (DDRBLOCK) or transaction spooling.

Column	Type	Description
ddr_state	char(24)	The current state of log capture: • Running = Log capture is running normally • Down = Log capture is not running • Uninitialized = The server is not a source server for replication
ddr_snoopy_loguniq	integer	The current log ID at which transactions are being captured for replication
ddr_snoopy_logpos	integer	The current log position at which transactions are being captured for replication
ddr_replay_loguniq	integer	The current log ID at which transactions have been applied
ddr_replay_logpos	integer	The current log position at which transactions have been applied. This is the position from which the log would need to be replayed to recover Enterprise Replication if Enterprise Replication or the database server shut down.
ddr_curr_loguniq	integer	The current log ID
ddr_curr_logpos	integer	The current log position
ddr_logsnoop_cached	integer	The number of log pages that log capture read from its cache
ddr_logsnoop_disk	integer	The number of times that log capture had to read log pages from disk
ddr_log_tossed	integer	The number of log pages that could not be stored in the cache because the log capture buffer cache was full
ddr_logs_ignored	integer	The number of log records that were ignored because they were extensible log records unknown to Enterprise Replication
ddr_dlog_requests	integer	The number of times that a dynamic log was requested to be created to prevent DDRBLOCK state
ddr_total_logspace	integer	The total number of log pages in the replication system
ddr_logspace2wrap	integer	The number of log spaces until log capture runs into a log wrap
ddr_logpage2block	integer	The number of log pages until log capture runs into a DDRBLOCK state
ddr_logneeds	integer	The number of log pages necessary to prevent a log wrap to avoid a DDRBLOCK state
ddr_logcatchup	integer	The number of log pages necessary to process before going out of a DDRBLOCK state
ddr_loglag_state	char(10)	The state of DDR log lag: on or off
ddr_cur_loglag_act	char(24)	The action being taken to prevent log wrapping
ddr_logstage_diskusage	float	The amount of used log staging disk space as a percentage of the total space
ddr_logstage_hwm4disk	integer	The maximum allowable disk space for log staging in KB

Column	Type	Description
ddr_logstage_maxused	float	The maximum disk space ever used for log staging in KB
ddr_logstage_lfile_cnt	integer	The number of staged log files

The syscdr_nif Table

The **syscdr_nif** table contains information about network connections and the flow of data between Enterprise Replication servers.

Column	Type	Description
nif_connid	integer	The CDRID of the peer node
nif_connname	char(24)	The name (group name) of the peer node
nif_state	char(24)	The status of the Enterprise Replication network: • Admin Close = Enterprise Replication was stopped by user by issuing the **cdr stop** command • Connected = The connection is active • Connecting = The connection is being established • Disconnected = The connection was explicitly disconnected • Local server = The connection is to the local server. • Logic Error = The connection disconnected due to an error during message transmission • Never Connected = The servers have never had an active connection • Start Error = The connection disconnected due to an error while starting a thread to receive remote messages • Timeout = The connection attempt has timed out, but will be reattempted
nif_connstate	char(24)	The connection state: • ABORT = The connection is being aborted. • BLOCK = The connection has been blocked from transmitting data by the other server. • INIT = The connection is being initialized. • INTR = The connection has been interrupted. • RUN = The connection is active. • SHUT = The connection is shutting down in an orderly way. • SLEEP = The connection is waiting to receive data. • STOP = The connection is stopped. • SUSPEND = The connection has been suspended. • TIMEOUT = The connection has timed out.
nif_version	integer	The network protocol of this connection used to convert the message formats between dissimilar releases of the server, for example, IBM Informix 7 and IBM Informix 9
nif_msgsent	integer	Number of messages sent to the peer server
nif_bytessent	integer	Number of bytes sent to the peer server
nif_msgrcv	integer	Number of messages received from the peer server
nif_bytesrcv	integer	Number of bytes received from the peer server

Column	Type	Description
nif_compress	integer	Compression level for communications • -1 = no compression • 0 = compress only if the target server expects compression • 1 - 9 = increasing levels of compression
nif_sentblockcnt	integer	Number of times a flow block request was sent to the peer server to delay sending any further replicated transactions for a short time because the receive queue on the target server is full
nif_rcvblockcnt	integer	Number of times a flow block request was received from the peer server
nif_trgsend_stamp1	integer	Stamp 1 of the last transaction sent to the peer server
nif_trgsend_stamp2	integer	Stamp 2 of the last transaction sent to the peer server
nif_acksend_stamp1	integer	Stamp 1 of the last acknowledgment sent to the peer server
nif_acksend_stamp2	integer	Stamp 2 of last acknowledgment sent to the peer server
nif_ctrlsend_stamp1	integer	Stamp 1 of the last control message sent to the peer server
nif_ctrlsend_stamp2	integer	Stamp 2 of the last control message sent to the peer server
nif_syncsend_stamp1	integer	Stamp 1 of the last sync message sent to the peer server
nif_syncsend_stamp2	integer	Stamp 2 of the last sync message sent to the peer server
nif_starttime	datetime	Time that the connection was established
nif_lastsend	datetime	Time of the last message sent to the peer server

The syscdr_rcv Table

The **syscdr_rcv** table contains information about transactions being applied on target servers and acknowledgments being sent from target servers.

Column	Type	Description
rm_state	char(100)	The status of the receive manager and apply threads: • Running = Transaction apply is running normally • Down = Transaction apply is not running • Uninitialized = The server is not a source server for replication
rm_num_sleepers	integer	Number of data sync threads currently suspended
rm__num_dsthreads	integer	The current number of data sync threads
rm_min_dsthreads	integer	Minimum number of data sync threads
rm_max_dsthreads	integer	Maximum number of data sync threads
rm_ds_block	integer	If 1, the data sync is currently blocked to try to avoid causing a DDRBLOCK state
rm_ds_parallel	integer	The degree to which transactions are applied in parallel (0 through 3, inclusive): • 0 = the highest degree of parallelism • 3 = serial apply (no parallelism)
rm_ds_failrate	float	A computed weighted ratio that is used to determine when to change the degree of apply parallelism based on the rate of transactions that could not be applied
rm_ds_numrun	integer	Number of transactions run
rm_ds_lockout	integer	Number of lock timeouts encountered

Column	Type	Description
rm_ds_lockrb	integer	Number of forced rollbacks due to having to switch to serial apply
rm_ds_num_deadlocks	integer	Number of deadlocks encountered
rm_ds_num_pcommits	integer	Number of out-of-order commits that have occurred
rm_ack_waiting	integer	Number of acknowledgments that are waiting for a log flush to return to the source server
rm_tosleep	integer	Total times that the data sync threads have become suspended
rm_sleeptime	integer	Total time that the data sync threads have been suspended
rm_workload	integer	The current workload
rm_optscale	integer	Factor determining how many data sync threads will be allowed per CPU VP
rm_min_fthreads	integer	Minimum acknowledgment threads
rm_max_fthreads	integer	Maximum acknowledgment threads
rm_ack_start	char(64)	Time when the acknowledgment threads started
rm_ds_start	char(64)	Time when the data sync threads started
rm_pending_acks	integer	Number of acknowledgments on the source that have not yet been processed
rm_blob_error_bufs	integer	Number of smart large objects that could not be successfully applied

The syscdr_ris Table

The **syscdr_ris** table contains the first ten lines of the transaction header for each RIS file.

Column	Type	Description
ris_rid	integer	Pseudo row ID.
ris_file	char(128)	RIS file name.
ris_sourceid	integer	CDRID of source server.
ris_source	char(128)	Source server name.
ris_committime	char(20)	Time when the transaction was committed on the source server.
ris_targetid	char(128)	CDRID of the target server.
ris_target	integer	Target server name.
ris_receivetime	char(20)	Time when the transaction was received on the target server.
ris_atsfile	char(128)	Corresponding ATS file.
ris_line1	char(200)	The first line of the transaction header information.
ats_line2	char(200)	The second line of the transaction header information.
ats_line3	char(200)	The third line of the transaction header information.
ats_line4	char(200)	The fourth line of the transaction header information.
ris_line5	char(200)	The fifth line of the transaction header information.
ris_line6	char(200)	The sixth line of the transaction header information.
ris_line7	char(200)	The seventh line of the transaction header information.
ris_line8	char(200)	The eighth line of the transaction header information.

Column	Type	Description
ris_line9	char(200)	The ninth line of the transaction header information.
ris_line10	char(200)	The tenth line of the transaction header information.

The syscdr_risdir Table

The **syscdr_risdir** table contains information about the contents of the RIS directory.

Column	Type	Description
risd_rid	integer	Pseudo row ID
risd_file	char(128)	RIS file name
risd_mode	integer	File mode
risd_size	integer	File size in bytes
risd_atime	datetime	Last access time
risd_mtime	datetime	Last modified time
risd_ctime	datetime	Create time

The syscdr_rqm Table

The **syscdr_rqm** table contains 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.

Column	Type	Description
rqm_idx	integer	Index number
rqm_name	char(128)	Queue name
rqm_flags	integer	Flags
rqm_txn	integer	Transactions in queue
rqm_event	integer	Events in queue
rqm_txn_in_memory	integer	Transaction in memory
rqm_txn_in_spool_only	integer	Spool-only transactions
rqm_txn_spooled	integer	Spooled transactions
rqm_unspooled_bytes	int8	Unspooled bytes
rqm_data_in_queue	int8	Data in queue
rqm_inuse_mem	int8	Real memory in use
rqm_pending_buffer	integer	Pending buffers
rqm_pending_data	int8	Pending buffers
rqm_maxmemdata	int8	Maximum memory in use by data
rqm_maxmemhdr	int8	Maximum memory in use by headers
rqm_totqueued	int8	Total data queued

Column	Type	Description
rqm_tottxn	integer	Total transactions queued
rqm_totspooled	integer	Total transactions spooled
rqm_totrestored	integer	Total transactions stored
rqm_totrecovered	integer	Total transactions recovered
rqm_totspoolread	integer	Total rows read from spool
rqm_totdeleted	integer	Total transactions deleted
rqm_totduplicated	integer	Total transactions duplicates
rqm_totlookup	integer	Total transaction lookups

The syscdr_rqmhandle Table

The **syscdr_rqmhandle** table contains information about which transaction is being processed in each queue. The handle marks the position of the thread in the queue.

Column	Type	Description
rqmh_qidx	integer	The queue associated with this handle
rqmh_thread	char(18)	Thread owning the handle
rqmh_stamp1	integer	Stamp 1 of the last transaction this handle accessed
rqmh_stamp2	integer	Stamp 2 of the last transaction this handle accessed
rqmh_servid	integer	Part 1 of the transaction key
rqmh_logid	integer	Part 2 of the transaction key
rqmh_logpos	integer	Part 3 of the transaction key
rqmh_seq	integer	Part 4 of the transaction key

The syscdr_rqmstamp Table

The **syscdr_rqmstamp** table contains information about which transaction is being added to each queue.

Column	Type	Description
rqms_qidx	integer	Queue index corresponding to the queues: • `0` = Transaction Send Queue • `1` = Acknowledgment Send Queue • `2` = Control Send Queue • `3` = CDR Metadata Sync Send Queue • `4` = Transaction Receive Queue
rqms_stamp1	integer	Stamp 1 of the next transaction being put into the queue
rqms_stamp2	integer	Stamp 2 of the next transaction being put into the queue
rqms_cstamp1	integer	Communal stamp 1 used to identify the next transaction read from the receive queue
rqms_cstamp2	integer	Communal stamp 2 used to identify the next transaction read from the receive queue

The syscdr_state Table

The **syscdr_state** table contains status on Enterprise Replication, data capture, data apply, and the network between the servers.

Column	Type	Description
er_state	char(24)	The status of Enterprise Replication: • Abort = Enterprise Replication is aborting on this server. • Active = Enterprise Replication is running normally. • Down = Enterprise Replication is stopped on this server. • Dropped = The attempt to drop the **syscdr** database failed. • Init Failed = The initial start-up of Enterprise Replication on this server failed, most likely because of a problem on the specified global catalog synchronization server. • Initializing = Enterprise Replication is being defined. • Initial Startup = Enterprise Replication is starting for the first time on this server. • Shutting Down = Enterprise Replication is shutting down on this server. • Startup Blocked = Enterprise Replication cannot start because the server was started with the **oninit -D** command. • Synchronizing Catalogs = The server is receiving a copy of the **syscdr** database. • Uninitialized = The server does not have Enterprise Replication defined on it.
er_capture_state	char(24)	The current state of log capture: • Running = Log capture is running normally • Down = Log capture is not running • Uninitialized = The server is not a source server for replication
er_network_state	char(64)	The status of the Enterprise Replication network: • Running = Communication is running normally • Down = Communication is not running • Uninitialized = The server is not a source server for replication
er_apply_state	char(24)	The status of the receive manager and apply threads: • Running = Transaction apply is running normally • Down = Transaction apply is not running • Uninitialized = The server is not a source server for replication

The syscdrack_buf Table

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 G-18.

The syscdrack_txn Table

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 G-17.

The syscdrctrl_buf Table

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 G-18.

The syscdrctrl_txn Table

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 G-17.

The syscdrerror Table

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

The syscdrlatency Table

The **syscdrlatency** table contains statistics about Enterprise Replication latency (the time it takes to replicate transactions).

Column	Type	Description
source	integer	Source of transaction (**cdrid**)
replid	integer	Replicate ID
txncnt	integer	The number of transactions on this source replicate

Column	Type	Description
inserts	integer	Number of INSERT statements
deletes	integer	Number of DELETE statements
updates	integer	Number of UPDATE statements
last_tgt_apply	integer	The time of the last transaction to be applied to the target (**cdrtime**)
last_src_apply	integer	The time of the last transaction to be applied on the source (**cdrtime**)

The syscdrpart Table

The **syscdrpart** 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
pendingsync	integer	• `0` = the Pending Sync attribute is not set • `1` = the Pending Sync attribute is set, indicating that the participant is waiting to be synchronized after the replication server was enabled

The syscdrprog Table

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
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

Column	Type	Description
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

The syscdrq Table

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

The syscdrqueued Table

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*

The syscdrrecv_buf Table

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 G-18.

The syscdrrecv_stats Table

The **syscdrrecv_stats** table contains statistics about the receive manager. The receive manager is a set of service routines between the receive queues and data sync.

Column	Type	Description
source	integer	The source server (**cdrid**)
txncnt	integer	Number of transactions from this source
pending	integer	The transaction currently pending on this source
active	integer	The transaction currently active on this source
maxpending	integer	Maximum pending transactions on this source
maxactive	integer	Maximum active transactions on this source
avg_pending	float	Average pending transactions on this source
avg_active	float	Average active transactions on this source
cmtrate	float	Average commit rate from this source

The syscdrrecv_txn Table

The **syscdrrecv_txn** table contains information about the data receive queue. The receive queue resides in memory.

When a replication server receives replicated data from a source database server, it puts the data in the receive queue and then applies the transactions on the target.

For information on the columns of the **syscdrrecv_txn** table, refer to "Columns of the Transaction Tables" on page G-17.

The syscdrrepl Table

The **syscdrrepl** table contains replicate information.

Column	Type	Description
replname	lvarchar	Replicate name.
replstate	char(50)	Replicate state. For possible values, refer to "cdr list server" on page A-103.
freqtype	char(1)	Type of replication frequency: • `C` = continuous • `I` = interval • `T` = time based • `M` = day of month • `W` = day of week
freqmin	smallint	The time for replication by minute: • Minutes after the hour that replication should occur. • Null if continuous.
freqhour	smallint	The time for replication by hour: • Hour that replication should occur. • Null if continuous.

Column	Type	Description
freqday	smallint	Day of week or month replication should occur.
scope	char(1)	Replication scope: • T = transaction • R = row-by-row
invokerowspool	char(1)	Whether Row Information Spooling is enabled: • Y = row spooling is enabled. • N = row spooling is disabled.
invoke transpool	char(1)	Whether Aborted Transaction Spooling is enabled: • Y = transaction spooling is enabled. • N = transaction spooling is disabled.
primresolution	char(1)	Type of primary conflict resolution: • A = always apply • D = delete wins • I = ignore • T = timestamp. • S = SPL routine
secresolution	char(1)	Type of secondary conflict resolution: • S = SPL routine • Null = not configured
storedprocname	lvarchar	SPL routine: • Name of SPL routine for secondary conflict resolution. • Null if not defined.
floattype	char(1)	Type of floating point number conversion: • 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)	Whether triggers are enabled: • Y = triggers are enabled. • N = triggers are disabled.
isfullrow	char(1)	Whether to replicate full rows or only the changed columns: • Y = sends the full row and enables upserts. • N = sends only changed columns and disables upserts.
isgrid	char(1)	Whether the replicate belongs to a grid replicate set: • Y = the replicate belongs to a grid replicate set. • N = the replicate does not belong to a grid replicate set.

Related tasks:

"Viewing grid information" on page 7-7

The syscdrreplset Table

The **syscdrreplset** table contains replicate set information.

Column	Type	Description
replname	lvarchar	Replicate name
replsetname	lvarchar	Replicate set name

Column	Type	Description
replsetattr	integer	Replicate set attributes: • 0x00200000U = The replicate set was created with a template. • 0x00000080U = The replicate set is exclusive.

The syscdrs Table

The **syscdrs** table contains information about database servers in an Enterprise Replication domain.

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) • E = Error during connection • F = Connection failed • K = In the process of connecting • L = The connection is to the local server • R = Disconnected but will attempt to reconnect • T = Idle time-out caused connection to terminate • X = Connection closed by user command and unavailable until reset by user
cnnstatechg	integer	Time that connection state was last changed.
servstate	char(1)	Status of database server: • A = Active. The server is active and replicating data. • D = Deleted. The server has been deleted; it is not capturing or delivering data and the queues are being drained. • S = Suspended. Delivery of replication data to the server is suspended. • Q = Quiescent. The server is in the process of being defined. • U = Disabled. Only delete shadow tables are populated in this state.
ishub	char(1)	Whether the server is a hub server that forwards information to another replication server: • Y = Server is a hub • N = Server is not a hub
isleaf	char(1)	Whether the server is a leaf or a nonleaf server: • 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	The number of minutes of idle time between replication servers before the connection is timed out.

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.

Related concepts:

"Hierarchical Routing Topology Terminology" on page 3-16

The syscdrsend_buf Table

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 G-18.

The syscdrsend_txn Table

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 G-17.

The syscdrserver Table

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 and unavailable until reset by user
connstatechange	integer	Time that connection state was last changed
servstate	char(50)	Status of this database server: • A = Active • D = Disabled • S = Suspended • Q = Quiescent (initial sync state only)

Column	Type	Description
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

The syscdrsync_buf Table

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 G-18.

The syscdrsync_txn Table

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 G-17.

The syscdrtx Table

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*
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

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

Column	Type	Description
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.

Appendix H. Replication Examples

This appendix contains simple examples of replication using the command-line utility (CLU).

Appendix A, "The cdr Command-Line Utility Reference," on page A-1 documents the CLU.

Related tasks:

"Preparing the Network Environment" on page 4-1

Replication Example Environment

To run the replication examples in this chapter, you need three IBM 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**. 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**.

For information about database server groups, see "Database Server Groups" on page 4-2.

UNIX Only

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_ital
g_japan  group     -      -          i=6
japan    ontlitcp  s3     techpubs6  g=g_japan
```

Windows Only

For more information about how to prepare the SQLHOSTS connectivity information using the information shown in the above UNIX**sqlhosts** file, see Appendix I, "SQLHOSTS Registry Key (Windows)," on page I-1.

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 4-10 and "CDR_QDATA_SBSPACE Configuration Parameter" on page B-12.

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 about preparing data for replication, see "Data Preparation Example" on page 4-26.

Primary-Target Example

This is 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.

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 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.

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
   ```

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 primary-target 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.

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 replicate 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     0
   ```

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 repl2
   ```

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
```

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

This example 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 H-1 on page H-7 shows the replication hierarchy.

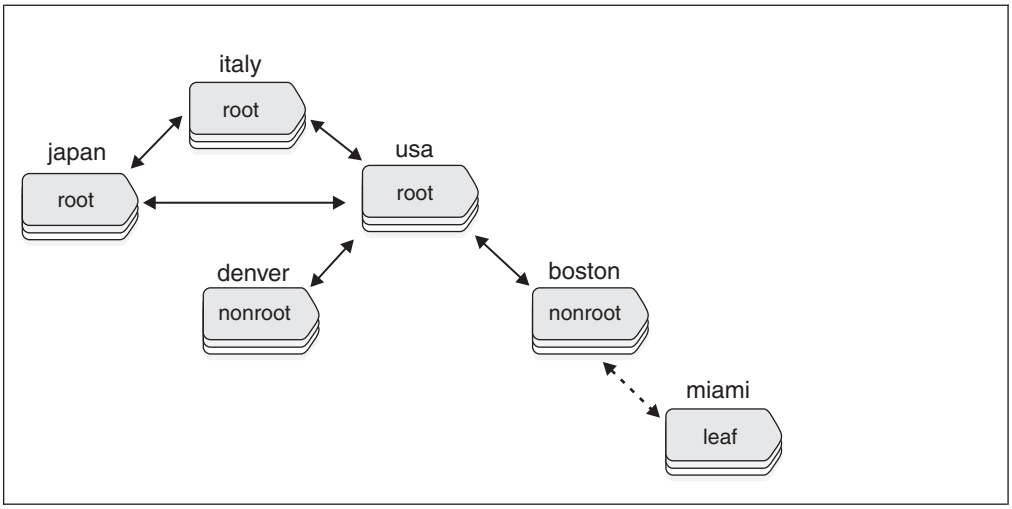


Figure H-1. Hierarchical Tree Example

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 H-1.

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-2.) 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
   ```

Appendix I. SQLHOSTS Registry Key (Windows)

You manage connectivity information on Windows operating systems with the SQLHOST 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 includes 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 INFORMIXSQLHOSTS 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 for each database server 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 I-3.
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 I-4.
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 I-4.

Setting up the SQLHOSTS Registry with ISA

It is strongly recommended that you use IBM Informix Server Administrator (ISA), rather than **regedit**, 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.

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 for Database Server with regedit

It is recommended that you use ISA to set up the SQLHOSTS registry key.

Important: Use extreme caution with **regedit**. If you make mistakes when editing the registry, you can destroy the configurations, not only of your IBM Informix products, but of your other applications.

To set up SQLHOSTS with regedit:

1. Run the Windows program, **regedit**.
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, add a new key.

5. Give the new key the name of the database server.
6. Select the new key that you just made (the key with the database server name) and add a new string value for it.
7. Give the value the name of one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, or SERVICE). Give the OPTIONS field the value of the database server group to which this database server will belong.
8. Modify the value to add value data.
9. Repeat steps 6 through 8 for each field of the **sqlhosts** information.

For example, a database server named **iris_112** could have a key under the SQLHOSTS key with the following values:

- HOST: iris
- OPTIONS: g=g_iris
- PROTOCOL: onsoctcp
- SERVICE: techpubs27

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.

In this manual, 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.

To set up the database server group registry key

1. With the SQLHOSTS key selected, choose to add a new key.
2. Give the new key the name of the database server group. This value must correspond to the OPTIONS value in the database server name key.
3. Select the key with the database server group name that you just created and add a new string value for it.
4. Give the value the name of one of the fields of the **sqlhosts** information (HOST, OPTIONS, PROTOCOL, SERVICE).
5. Add a value for the field. For a database server group, the **sqlhosts** information fields should have 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.
6. Repeat steps 4 and 5 for the remaining fields of the **sqlhosts**
7. Select the database server group key and choose to add a key.
8. Give the new key the name of the database server. This value must correspond to the database server key, whose OPTIONS value was set to the database server group key.
9. If you are combining Enterprise Replication with HDR, create keys for primary and secondary HDR servers under the same database server group.
10. Exit from the Registry Editor.

For example, a database server group named **g_iris** could have a key under the SQLHOSTS key with the following values:

- HOST: -
- OPTIONS: i=5327
- PROTOCOL: group
- SERVICE: -

A key for the database server **iris_112** would appear both directly under SQLHOSTS and under **g_iris**:

```
SQLHOSTS
 iris_112
 g_iris
  iris_112
```

Related concepts:

"Database Server Groups" on page 4-2

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 for Database Server with regedit" on page I-2.
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 I-3.

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:\Windows/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 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
   ```

Appendix J. Data Sync Warning and Error Messages

This topic lists data sync warning and error messages that you can suppress from being written to the ATS and RIS files.

You cannot suppress code 0, DSROWCOMMITTED, which indicates that the row was committed, or code 1, DSEROW, which indicates that an error occurred.

To specify which warnings and errors to suppress, use the CDR_SUPPRESS_ATSRISWARN configuration parameter. For more information, see "CDR_SUPPRESS_ATSRISWARN Configuration Parameter" on page B-14.

Table J-1. Data sync warning and error messages

Warning or Error Code	Number	Description
DSEReplInsertOrder	2	Warning: Insert exception, row already exists in target table, converted to update
DSEReplUpdateOrder	3	Warning: Update exception, row does not exist in target table, converted to insert
DSEReplDeleteOrder	4	Warning: Delete exception, row does not exist in target table, saved in delete table
DSEReplInsert	5	Error: Insert aborted, row already exists in target table
DSEReplUpdate	6	Error: Update aborted, row does not exist in target table
DSEReplDelete	7	Error: Delete aborted, row does not exist in target table
DSERowLength	8	Error: Length of replicated row is greater than row size in target table
DSEDbopType	9	Error: Unknown db operation
DSENoServerTimeCol	10	Error: Missing cdrserver and/or cdrtime columns in target table
DSEConflictRule	13	Error: Unknown conflict resolution rule defined
DSELostConflictRes	14	Error: Failed conflict resolution rule
DSENoServerName	15	Error: Global catalog cannot translate replicate server id to name
DSEColMap	16	Error: Unable to remap columns selected for replication
DSEColUncomp	17	Error: Invalid char/length in VARCHAR column
DSESPRetTypeOp	18	Error: Invalid data type or unknown operation returned by stored procedure
DSESPAbortRow	19	Error: Row aborted by stored procedure
DSESPSelCols	20	Error: Number of columns returned by stored procedure not equal to the number of columns in select statement
DSESPColTypeLen	21	Error: Invalid data type or length for selected columns returned by stored procedure
DSESPError	22	Error: Error returned by user's stored procedure
DSESPInternal	23	Error: Internal error (buffer too small for stored procedure arguments
DSENoMatchKeyInsert	24	Error: No matching key delete row for key insert
DSESql	25	Error: SQL error encountered
DSEIsam	26	Error: ISAM error encountered

Table J-1. Data sync warning and error messages (continued)

Warning or Error Code	Number	Description
DSELocalDReExist	27	Warning: Local delete row has been reinserted on local server
DSELocalDOddState	28	Warning: Unable to determine if the local delete row should be updated to the delete table
DSELocalDInDelTab	29	Warning: Row already exists in delete table for the given local delete row
DSEBlobOrder	30	Warning: Row failed conflict resolution rule but one or more blob columns were accepted
DSEBlobSetToNull	31	Warning: One or more blob columns were set to NULL because data could not be sent
DSEBlobKeepLocal	32	Warning: One or more blob columns were not changed because data could not be sent
DSEBlobInvalidFlag	33	Error: Invalid user action defined for blob columns
DSEBlobAbortRow	34	Error: Row aborted by user's stored procedure due to unsent blobs
DSESPBlobRetOp	35	Error: Invalid action returned by user's stored procedure on blob columns
DSEReplDel	36	
DSENoUDTHeader	37	
DSENoUDTTrailer	38	
DSEStreamHandle	39	
DSEAttachUDREnv	40	
DSECdrreceiveSetup	41	
DSECdrreceiveCall	42	
DSECdrreceiveRetCode	43	cdrreceive returned error
DSECdrreceiveRetGarbage	44	cdrreceive returned garbage
DSEStream	45	Error reading from stream
DSEStreamAborted	46	Stream aborted by sender
DSEValStore	47	
DSECdrreceiveRetType	48	cdrreceive returned wrong type
DSEStreamOptType	49	Unrecognized stream option
DSEStreamOptLen	50	Stream option has bad length
DSEStreamOptBitmap	51	Error in changed col bitmap
DSEUnStreamColl	52	Error while unstreaming collection
DSEUnStreamRowType	53	Error while unstreaming rowtype
DSEStreamFormat	54	Unexpected or invalid data in stream
DSEStack	55	Out of stack space
DSEInternal	56	Generic internal problem
DSESmartBlobCreate	57	Error creating sblob
DSESmartBlobWrite	58	Error writing sblob
DSEStreamColConv	59	Error converting column data from the master dictionary formats to the local dictionary format
DSE2UTF8CodeSetConvErr	63	Error converting data from local database code set to UTF-8
DSEFromUTF8CodeSetConvErr	64	Error converting data from UTF-8 to local database code set.

Table J-1. Data sync warning and error messages (continued)

Warning or Error Code	Number	Description
DSE2UTF8CodeSetConvWarn	65	One or more characters were substituted during conversion from the local database code set to UTF-8.
DSEFromUTF8CodeSetConvWarn	66	One or more characters were substituted during conversion from UTF-8 to the local database code set.

Appendix K. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix products

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility features

The following list includes the major accessibility features in IBM Informix products. These features support:

- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Keyboard navigation

This product uses standard Microsoft Windows navigation keys.

Related accessibility information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software.

IBM and accessibility

See the *IBM Accessibility Center* at http://www.ibm.com/able for more information about the IBM commitment to accessibility.

Dotted decimal syntax diagrams

The syntax diagrams in our publications are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader.

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, that element is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 refers to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

? Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

***** Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be

repeated. For example, if you hear the line `5.1* data-area`, you know that you can include more than one data area or you can include none. If you hear the lines `3*, 3 HOST`, and `3 STATE`, you know that you can include `HOST`, `STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write `HOST STATE`, but you cannot write `HOST HOST`.
3. The `*` symbol is equivalent to a loop-back line in a railroad syntax diagram.

`+` Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line `6.1+ data-area`, you must include at least one data area. If you hear the lines `2+`, `2 HOST`, and `2 STATE`, you know that you must include `HOST`, `STATE`, or both. As for the * symbol, you can repeat a particular item if it is the only item with that dotted decimal number. The `+` symbol, like the `*` symbol, is equivalent to a loop-back line in a railroad syntax diagram.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy,

modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Index

Special characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Printed in USA

SC27-3539-04

Spine information:

Informix Product Family Informix **Version 11.70**

IBM Informix Enterprise Replication Guide