TimesTen to TimesTen Replication Guide

Release 7.0

B31684-03

ORACLE

TIMESTEN

September 2007

Printed in the United States of America

Contents

About this Guide

1 TimesTen Replication

2 Quick Start

3 Defining Replication Schemes

4 Setting Up a Replicated System

5 Monitoring Replication

6 Altering Replication

7 Administering an Active Standby Pair

8 Conflict Resolution and Failure Recovery

9 XML Document Type Definition for the Conflict Report File

Glossary

Index

About this Guide

This guide is for application developers and for system administrators who configure and manage TimesTen to TimesTen Replication. It provides:

- Background information on how TimesTen Replication works.
- Procedures and examples for common replication tasks.

To work with this guide, you should understand how database systems work. You should also have knowledge of SQL (Structured Query Language) and either ODBC (Open DataBase Connectivity) or JDBC (Java DataBase Connectivity). See "Background reading" on page 4 if you are not familiar with these interfaces.

TimesTen documentation

TimesTen documentation is available on the product distribution media and on the Oracle Technology Network: http://www.oracle.com/technology/documentation/timesten_doc.html.

Including this guide, the TimesTen documentation set consists of these documents:

Book Titles	Description
Oracle TimesTen In-Memory Database Installation Guide	Contains information needed to install and configure TimesTen on all supported platforms.
Oracle TimesTen In-Memory Database Introduction	Describes all the available features in the Oracle TimesTen In-Memory Database.
Oracle TimesTen In-Memory Database Operations Guide	Provides information on configuring TimesTen and using the ttIsql utility to manage a data store. This guide also provides a basic tutorial for TimesTen.
Oracle TimesTen In-Memory Database C Developer's and Reference Guide and the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide*	Provide information on how to use the full set of available features in TimesTen to develop and implement applications that use TimesTen.
Oracle TimesTen In-Memory Database API Reference Guide	Describes all TimesTen utilities, procedures, APIs and provides a reference to other features of TimesTen.
Oracle TimesTen In-Memory Database SQL Reference Guide	Contains a complete reference to all TimesTen SQL statements, expressions and functions, including TimesTen SQL extensions.
Oracle TimesTen In-Memory Database Error Messages and SNMP Traps	Contains a complete reference to the TimesTen error messages and information on using SNMP Traps with TimesTen.
Oracle TimesTen In-Memory Database TTClasses Guide	Describes how to use the TTClasses C++ API to use the features available in TimesTen to develop and implement applications.

TimesTen to TimesTen Replication Guide	Provides information to help you understand how TimesTen Replication works and step-by-step instructions and examples that show how to perform the most commonly needed tasks. This guide is for application developers who use and administer TimesTen and for system administrators who configure and manage TimesTen Replication.
TimesTen Cache Connect to Oracle Guide	Describes how to use Cache Connect to cache Oracle data in TimesTen data stores. This guide is for developers who use and administer TimesTen for caching Oracle data.
Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide	Provides information and solutions for handling problems that may arise while developing applications that work with TimesTen, or while configuring or managing TimesTen.

Background reading

For a Java reference, see:

- Horstmann, Cay and Gary Cornell. *Core Java(TM) 2, Volume I--Fundamentals (7th Edition) (Core Java 2)*. Prentice Hall PTR; 7 edition (August 17, 2004).

A list of books about ODBC and SQL is in the Microsoft ODBC manual included in your developer's kit. Your developer's kit includes the appropriate ODBC manual for your platform:

WINDOWS

- *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide* provides all relevant information on ODBC for Windows developers.

UNIX

- *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide*, included online in PDF format, provides information on ODBC for UNIX developers.

For a conceptual overview and programming how-to of ODBC, see:

- Kyle Geiger. *Inside ODBC*. Redmond, WA: Microsoft Press. 1995.

For a review of SQL, see:

- Melton, Jim and Simon, Alan R. *Understanding the New SQL: A Complete Guide*. San Francisco, CA: Morgan Kaufmann Publishers. 1993.
- Groff, James R. / Weinberg, Paul N. *SQL: The Complete Reference, Second Edition.* McGraw-Hill Osborne Media. 2002.

For information about Unicode, see:

- The Unicode Consortium, *The Unicode Standard, Version 5.0*, Addison-Wesley Professional, 2006.
- The Unicode Consortium Home Page at http://www.unicode.org

Conventions used in this guide

TimesTen supports multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to Windows 2000, Windows XP and Windows Server 2003. The term UNIX refers to Solaris, Linux, HP-UX, Tru64 and AIX.

TimesTen documentation uses these typographical conventions:

If you see...	It means...
`code font`	Code examples, filenames, and pathnames. For example, the `.odbc.ini.` or `ttconnect.ini` file.
`italic code font`	A variable in a code example that you must replace. For example: `Driver=`*`install_dir`*`/lib/libtten.sl` Replace *`install_dir`* with the path of your TimesTen installation directory.

TimesTen documentation uses these conventions in command line examples and descriptions:

If you see...	It means...
`fixed width italics`	Variable; must be replaced with an appropriate value. In some cases, such as for parameter values in built-in procedures, you may need to single quote (' ') the value.
`[ ]`	Square brackets indicate that an item in a command line is optional.
`{ }`	Curly braces indicated that you must choose one of the items separated by a vertical bar (|) in a command line.
`|`	A vertical bar (or pipe) separates arguments that you may use more than one argument on a single command line.
`...`	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
`%`	The percent sign indicates the UNIX shell prompt.
`#`	The number (or pound) sign indicates the UNIX root prompt.

TimesTen documentation uses these variables to identify path, file and user names:

If you see...	It means...
install_dir	The path that represents the directory where the current release of TimesTen is installed.
TTinstance	The instance name for your specific installation of TimesTen. Each installation of TimesTen must be identified at install time with a unique alphanumeric instance name. This name appears in the install path. The instance name “giraffe” is used in examples in this guide.
bits or *bb*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
release or *rr*	Two digits that represent the first two digits of the current TimesTen release number, with or without a dot. For example, 51 or 7.0 represents TimesTen Release 7.0.
jdk_version	Two digits that represent the version number of the major JDK release. Specifically, 14 represent JDK 1.4; 5 represents JDK 5.
`timesten`	A sample name for the TimesTen instance administrator. You can use any legal user name as the TimesTen administrator. On Windows, the TimesTen instance administrator must be a member of the Administrators group. Each TimesTen instance can have a unique instance administrator name.
DSN	The data source name.

Technical Support

For information about obtaining technical support for TimesTen products, go to the following Web address:

http://www.oracle.com/support/contact.html

1

TimesTen Replication

This chapter provides an overview of TimesTen replication. The general topics are:

- What is replication?
- How replication works
- Master/subscriber relationships
- Cache groups and replication
- Sequences and replication
- Foreign keys and replication
- Sequences and replication
- Replication schemes

What is replication?

Replication is the process of maintaining copies of data in multiple data stores. The purpose of TimesTen replication is to make data continuously available to mission-critical applications with minimal impact on performance.

Some of the benefits of replication include:

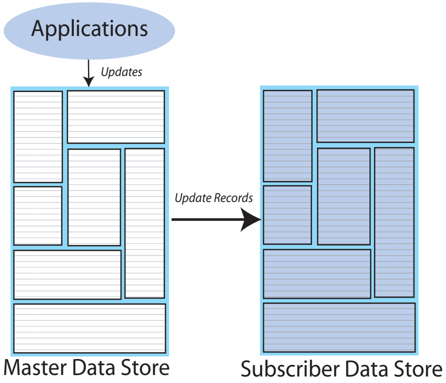
- Recovery from failures: You can maintain duplicate data stores on two or more servers. In the event of a software or hardware failure on one server, the data is available from data stores on other servers.
- Online upgrades and maintenance: When data is duplicated across data stores on different servers, you can perform upgrades, schema changes and other maintenance activities on one server, while providing your applications with continuous access to the replicated data on another server.
- Load sharing: By maintaining duplicate data stores on different servers, you can scale the number of servers to distribute workloads.

Master and subscriber data stores and elements

An entity that is replicated between data stores is called a *replication element*. TimesTen supports data stores, tables, and sequences as replication elements. As shown in Figure 1.1, TimesTen replication copies updates made to a *master* data store into a corresponding *subscriber* data store. You can replicate your entire data store or any number of selected tables to one or more subscriber data stores.

Currently, the master and subscriber data stores must reside on machines that have the same operating system, CPU type, and word size. Though you can replicate between data stores that reside on the same machine, replication is generally used for copying updates into a data store that resides on another machine. This helps prevent data loss from node failure. See "Configuring the network" on page 100 for information on the network requirements for replicating data between TimesTen systems.

Figure 1.1 Replicating the entire master data store



Requirements for replication compatibility

For replication to succeed between two data stores, the stores must be replication compatible. Two data stores are guaranteed to be replication compatible when their DSNs are configured with identical **DatabaseCharacterSet** and **TypeMode** attributes.

Note: If replication is configured between a data store from the current release of TimesTen and a data store from a TimesTen release previous to 7.0, then there are additional restrictions for replication compatibility. A data store may only replicate to a TimesTen release previous to 7.0 if it is configured with a **DatabaseCharacterSet** attribute of TIMESTEN8 and may only replicate tables with columns that use the original TimesTen data types (data types with the prefix TT_- or the data types BINARY_FLOAT and BINARY_DOUBLE). See "Types supported for backward compatibility in Oracle type mode" in the *Oracle TimesTen In-Memory Database SQL Reference Guide* for more information.

How replication works

This section describes the TimesTen replication agents and how they work together to replicate data from a master data store to its subscriber data stores.

Replication agents

Replication at each master and subscriber data store is controlled by a *replication agent*. The replication agent on the master data store reads the records from the transaction log and forwards any detected changes to replicated elements to the replication agent on the subscriber data store. The replication agent on the subscriber then applies the updates to its data store. If the subscriber agent is not running when the updates are forwarded by the master, the master retains the updates in the log until they can be transmitted.

The master and subscriber agents communicate through TCP/IP stream sockets. Each master and subscriber data store is identified by:

- A data store name derived from the file system's path name for the data store
- A host name

The replication agents obtain the TCP/IP address, host name, and other configuration information from the TTREP system tables described in Chapter 7, "System and Replication Tables" in the *Oracle TimesTen In-Memory Database SQL Reference Guide*.

How replication agents copy updates between data stores

In default replication, updates are copied between data stores in an asynchronous manner. Though asynchronous replication provides the best performance, it does not provide the application with confirmation that the replicated updates have been committed on the subscriber data stores. For “pessimistic” applications that need higher levels of confidence that the replicated data is consistent between the master and subscriber data stores, you can enable either the optional *return receipt* or *return twosafe* service.

The return receipt service loosely couples or “synchronizes” the application with the replication mechanism by blocking the application until replication confirms that the update has been received by the subscriber. The return twosafe service provides a fully synchronous option by blocking the application until replication confirms that the update has been both received and committed on the subscriber.

Return receipt replication has less performance impact than return twosafe at the expense of less synchronization. The operational details for asynchronous, return receipt, and return twosafe replication are discussed in the following sections:

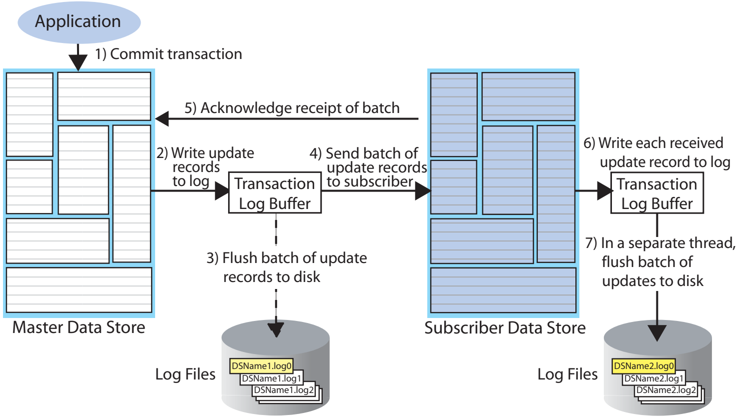
- Default replication
- Return receipt replication
- Return twosafe replication

Default replication

When using default TimesTen replication, an application updates a master data store and continues working without waiting for the updates to be received and applied by the subscribers. The master and subscriber data stores have internal mechanisms to confirm that the updates have been successfully received and committed by the subscriber. These mechanisms ensure that updates are applied at a subscriber only once, but they are completely independent of the application.

Default TimesTen replication provides maximum performance, but the application is completely decoupled from the receipt process of the replicated elements on the subscriber.

Figure 1.2 Basic asynchronous replication cycle


Application
1) Commit transaction
5) Acknowledge receipt of batch
2) Write update records to log
Transaction Log Buffer
4) Send batch of update records to subscriber
6) Write each received update record to log
Transaction Log Buffer
3) Flush batch of update records to disk
7) In a separate thread, flush batch of updates to disk
Master Data Store
Subscriber Data Store
Log Files
DSName1.log0
DSName1.log1
DSName1.log2
Log Files
DSName2.log0
DSName2.log1
DSName2.log2

The default TimesTen replication cycle is:

1. The application commits a local transaction to the master data store and is free to continue with other transactions.
2. During the commit, TimesTen Data Manager writes the transaction update records to the transaction log buffer.
3. The replication agent on the master data store directs the Data Manager to flush a batch of update records for the committed transactions from the log buffer to a log file on disk and synchronizes the disk. This step ensures that, if the master fails and you need to recover the data store from the checkpoint and log files on disk, the recovered master contains all the data it replicated to the subscriber.

   However, this flush-log-to-disk operation is skipped under any of the following conditions:

   - The update records for the committed transactions are already on disk.
   - The replication scheme is configured with a TRANSMIT NONDURABLE option, as described in "Setting transmit durability on data store elements" on page 49.
4. The master replication agent forwards the batch of transaction update records to the subscriber replication agent, which applies them to the subscriber data store.

Note: Update records are flushed to disk and forwarded to the subscriber in batches of 256K or less, depending on the master data store's transaction load. A batch is created when there is no more log data in the transaction log buffer or when the current batch is roughly 256K bytes. Batches are smaller than 256K bytes for lighter transaction loads when fewer log records are written.

5. The subscriber replication agent sends an acknowledgement back to the master replication agent that the batch of update records was received. (This acknowledgement includes information on which batch of records the subscriber last flushed to disk.) The master replication agent is now free to purge from the transaction log the update records that have been received, applied, and flushed to disk by all subscribers and to forward another batch of update records, while the subscriber replication agent asynchronously continues on to Step 6.
6. The replication agent at the subscriber updates the data store and directs its Data Manager to write the transaction update records to the transaction log buffer.
7. The replication agent at the subscriber data store uses a separate thread to direct the Data Manager to flush the update records to a disk-based log file.

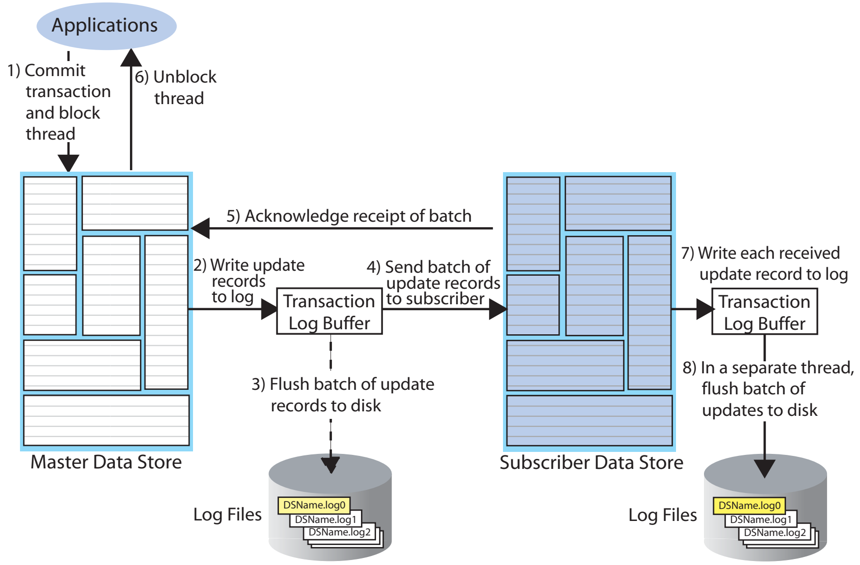
Return receipt replication

The return receipt service provides a level of synchronization between the master and a subscriber data store by blocking the application after commit on the master until the updates of the committed transaction have been received by the subscriber.

An application requesting return receipt updates the master data store in the same manner as in the basic asynchronous case. However, when the application commits a transaction that updates a replicated element, the master data store blocks the application until it receives confirmation that the updates for the completed transaction have been received by the subscriber.

Return receipt replication trades some performance in order to provide more “pessimistic” applications with the ability to ensure higher levels of data integrity and consistency between the master and subscriber data stores. In the event of a master failure, the application has a high degree of confidence that a transaction committed at the master persists in the subscribing data store.

Figure 1.3 Return receipt replication


Applications
1) Commit transaction and block thread
6) Unblock thread
5) Acknowledge receipt of batch
2) Write update records to log
Transaction Log Buffer
4) Send batch of update records to subscriber
7) Write each received update record to log
Transaction Log Buffer
3) Flush batch of update records to disk
8) In a separate thread, flush batch of updates to disk
Master Data Store
Subscriber Data Store
Log Files
DSName.log0
DSName.log1
DSName.log2
Log Files
DSName.log0
DSName.log1
DSName.log2

As shown in Figure 1.3, the return receipt replication cycle is the same as shown for the basic asynchronous cycle in Figure 1.2, only the master replication agent blocks the application thread after it commits a transaction (Step 1) and retains control of the thread until the subscriber acknowledges receipt of the update batch (Step 5). Upon receiving the return receipt acknowledgement from the subscriber, the master replication agent returns control of the thread to the application (Step 6), freeing it to continue executing transactions.

Note: In order to obtain the best compromise between assuring data store integrity and performance, return receipt is not fully synchronous. Though this service informs the application that the transaction has been received by the subscriber, it does not *guarantee* that the transaction has been committed or made durable in the subscriber data store.

If the subscriber is unable to acknowledge receipt of the transaction within a configurable timeout period (default is 10 seconds), the master replication agent returns a warning stating that it did not receive acknowledgement of the update from the subscriber and returns control of the thread to the application. The application is then free to commit another transaction to the master, which continues replication to the subscriber as before. Return receipt transactions may timeout for many reasons. The most likely causes for timeout are the network, a failed replication agent, or the master replication agent may be so far behind with respect to the transaction load that it cannot replicate the return receipt transaction before its timeout expires. For information on how to manage return-receipt timeouts, see "Managing return service timeout errors and replication state changes" on page 66.

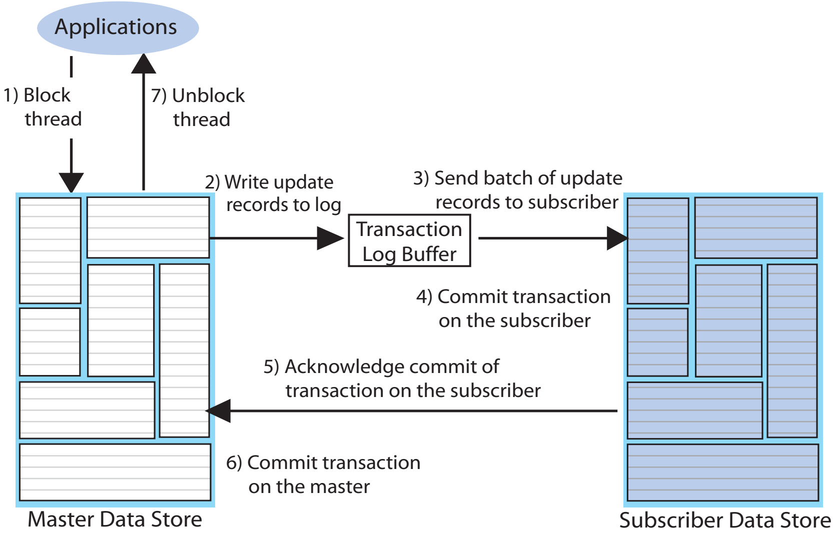
See "RETURN RECEIPT" on page 58 for information on how to configure replication for return receipt.

Return twosafe replication

The return twosafe service provides fully synchronous replication between the master and subscriber. Unlike the previously described replication modes, where transactions are transmitted to the subscriber after being committed on the master, transactions in twosafe mode are first committed on the subscriber before they are committed on the master.

Note: The return twosafe service can only be used in a "hot standby" replication scheme where there is a single master and subscriber and the replication element is the entire data store. See "General workload configuration" on page 20 for more information on the hot standby configuration.

Figure 1.4 Twosafe replication


Applications
1) Block thread
7) Unblock thread
2) Write update records to log
3) Send batch of update records to subscriber
Transaction Log Buffer
4) Commit transaction on the subscriber
5) Acknowledge commit of transaction on the subscriber
6) Commit transaction on the master
Master Data Store
Subscriber Data Store

The following describes the replication behavior between a master and subscriber configured for return twosafe replication:

1. The application commits the transaction on the master data store.
2. The master replication agent writes the transaction records to the log and inserts a special *precommit* log record before the commit record. This precommit record acts as a place holder in the log until the master replication receives an acknowledgement that indicates the status of the commit on the subscriber.

Note: Transmission of return twosafe transactions are nondurable, so the master replication agent does not flush the log records to disk before sending them to the subscriber, as it does by default when replication is configured for asynchronous or return receipt replication.

3. The master replication agent transmits the batch of update records to the subscriber.
4. The subscriber replication agent commits the transaction on the subscriber data store.
5. The subscriber replication agent returns an acknowledgement back to the master replication agent with notification of whether the transaction was committed on the subscriber and whether the commit was successful.
6. If the commit on the subscriber was successful, the master replication agent commits the transaction on the master data store.
7. The master replication agent returns control to the application.

   If the subscriber is unable to acknowledge commit of the transaction within a configurable timeout period (default is 10 seconds) or if the acknowledgement from the subscriber indicates the commit was unsuccessful, the replication agent returns control to the application without committing the transaction on the master data store. The application can then to decide whether to unconditionally commit or retry the commit. You can optionally configure your replication scheme to direct the master replication agent to commit all transactions that time out.

   See "RETURN TWOSAFE" on page 60 for information on how to configure replication for return twosafe.

Note: RETURN RECEIPT and RETURN TWOSAFE subscriber attributes cannot coexist. Transactions can be either be return twosafe or return receipt, but not both.

Master/subscriber relationships

You create a *replication scheme* to define a specific configuration of master and subscriber data stores. This section describes the possible relationships you can define between master and subscriber data stores when creating your scheme. How to create a replication scheme is described in "Replication schemes" on page 31.

When defining a relationship between a master and subscriber, you must consider some combination of the following:

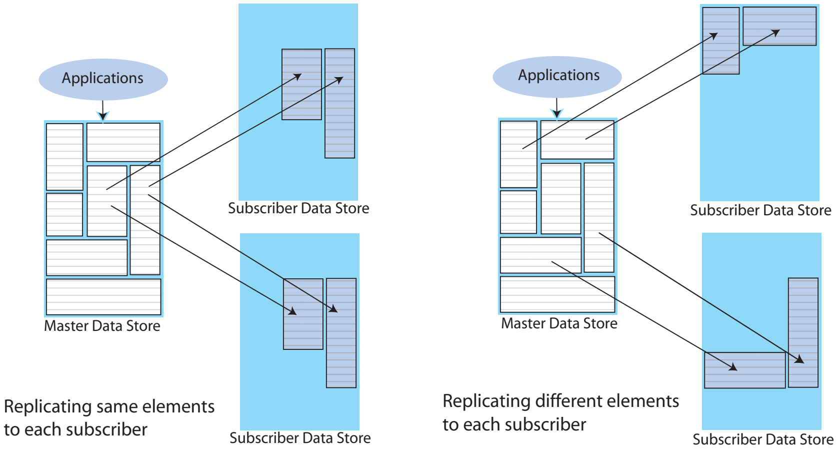
- Full or selective replication
- Unidirectional or bidirectional replication
- Direct replication or propagation
- Active standby pair with read-only subscribers

Full or selective replication

Figure 1.1 illustrates a *full* replication scheme in which the entire master data store is replicated to the subscriber. You can also configure your master and subscriber data stores in various combinations to selectively replicate some table elements in a master data store to subscribers.

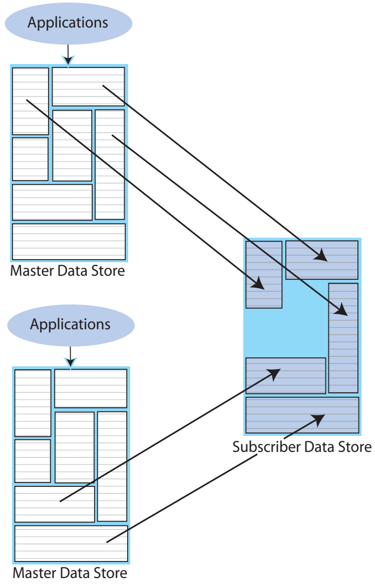
Figure 1.5 shows some examples of *selective* replication. The left side shows a master data store that replicates the same selected elements to multiple subscribers, while the right side shows a master that replicates different elements to each subscriber.

Figure 1.5 Replicating selected elements to multiple subscribers



Another way to use selective replication is to configure multiple master data stores to replicate elements to a single subscriber that serves as a common backup data store, as shown in Figure 1.6.

Figure 1.6 Multiple masters replicating to a single subscriber



Unidirectional or bidirectional replication

So far in this chapter, we have described *unidirectional replication*, where a master data store sends updates to one or more subscriber data stores. However, you can also configure data stores to operate *bidirectionally*, where each store is both a master and a subscriber.

There are two basic ways to use bidirectional replication:

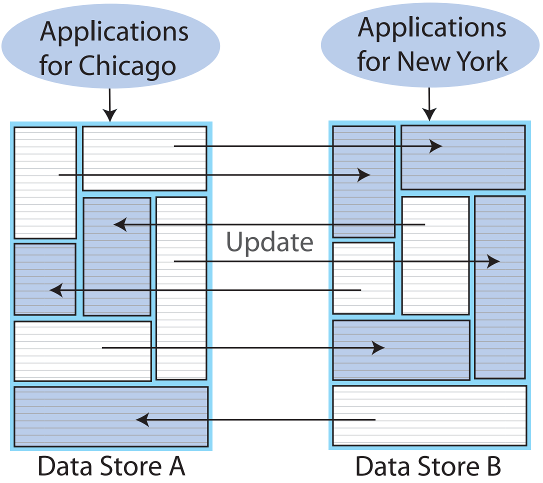
- Split workload configuration
- General workload configuration

Split workload configuration

In a *split workload* configuration, each data store serves as a master for some table elements and a subscriber for others.

Consider the example shown in Figure 1.7, where the accounts for Chicago are processed on data store A while the accounts for New York are processed on data store B.

Figure 1.7 “Split workload” bidirectional replication



Note: It may be difficult to achieve a clean split of the workload. In Figure 1.7, imagine that there are rows that must be updated by transactions on both Chicago and New York applications. In that case, update conflicts are possible in the shared rows.

General workload configuration

In a *general workload* configuration, each data store serves as both a master and subscriber for the same table elements. Applications on either data store can update any of the elements and the update is replicated to the other data store. This type of configuration is often referred to as *multimaster.*

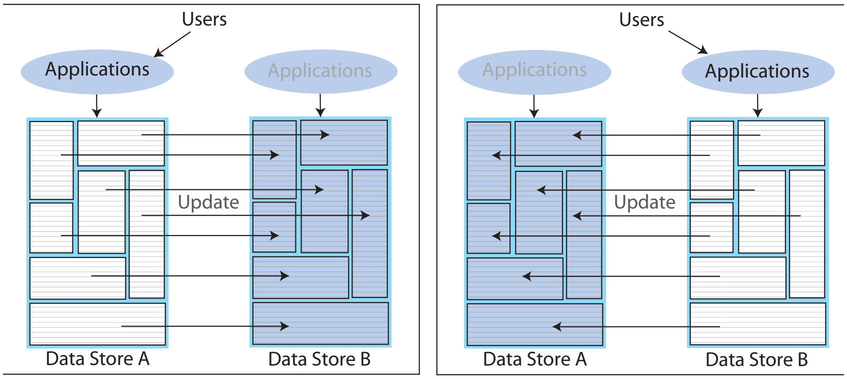
There are two basic types of general workload schemes:

- *Hot standby*: In this scheme, users access a specific application/data store combination that replicates updates to a duplicate backup application/data store combination. In the event of a failure, the user load can be quickly shifted to the backup application/data store.
- *Distributed workload*: In this scheme, user access is distributed across duplicate application/data store combinations that replicate any update on any element to each other. In the event of a failure, the affected users can be quickly shifted to any application/data store combination.

The hot standby configuration is shown in Figure 1.8. This configuration mimics the simplicity of unidirectional replication while allowing for simple and fast recovery in the event of a data store failure. Although there are two master data stores, applications update only one data store until it fails, at which time the applications are shifted to the other data store.

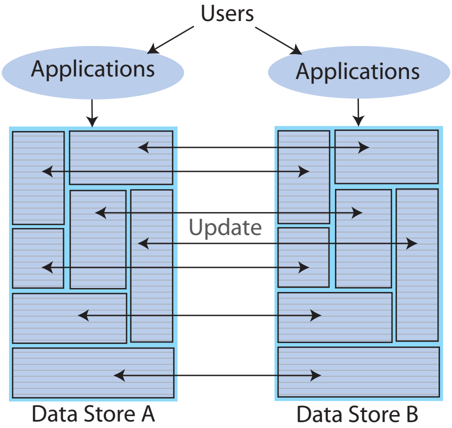
Users operate on data store A and updates are replicated to data store B, which assumes the role of subscriber. In the event data store A fails, users can be redirected to a copy of the application already configured on data store B. When data store A is restored, it can then assume the role of subscriber.

Figure 1.8 Hot standby configuration



The distributed workload configuration is shown in Figure 1.9. Users access duplicate applications on each data store, which serves as both master and subscriber for the other data store.

Figure 1.9 Distributed workload configuration



When data stores are replicated in a distributed workload configuration, it is possible for separate users to concurrently update the same rows and replicate the updates to one another. Your application should ensure that such conflicts cannot occur, that they be acceptable if they do occur, or that they can be successfully resolved using the conflict resolution mechanism described in "Replication conflict detection and resolution" on page 167.

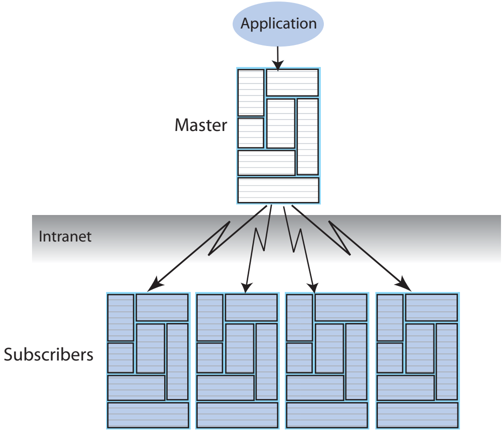
Note: A distributed workload configuration should not be used with the return twosafe return service, nor should it be used with Cache Connect to Oracle.

Direct replication or propagation

You can define a subscriber to serve as a *propagator* that receives replicated updates from a master and passes them on to subscribers of its own.

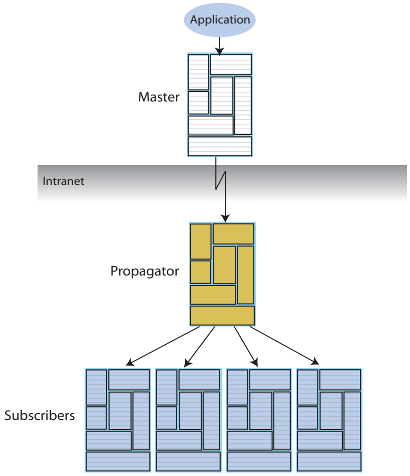
Propagators are useful for optimizing replication performance over lower-bandwidth network connections, such as those between servers in an intranet. For example, consider the *direct replication* configuration illustrated in Figure 1.10, where a master directly replicates to four subscribers over an intranet connection. Replicating to each subscriber over a network connection in this manner is an inefficient use of network bandwidth.

Figure 1.10 Master replicating to multiple subscribers over a network



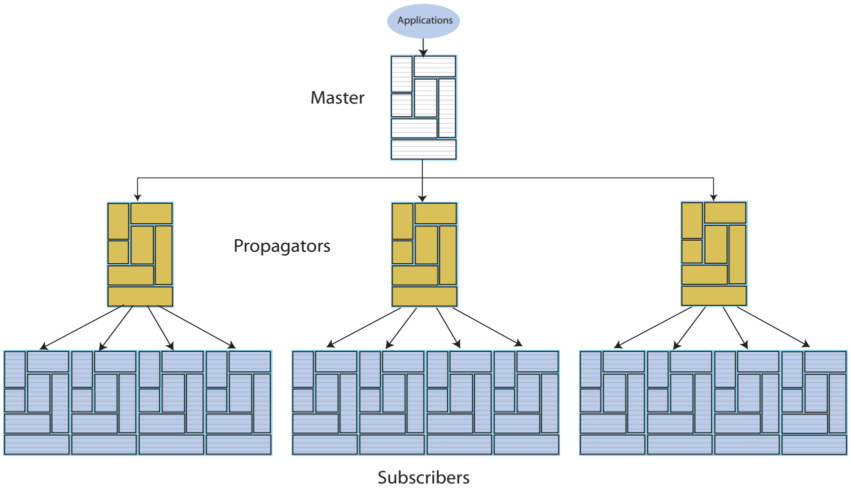
For optimum performance, consider the configuration shown in Figure 1.11, where the master replicates to a single propagator over the network connection. The propagator in turn forwards the updates to each subscriber on its local area network.

Figure 1.11 Master replicating to a single propagator over a network



Propagators are also useful for distributing replication loads in configurations that involve a master data store that must replicate to a large number of subscribers. For example, it is more efficient for the master to replicate to three propagators, rather than directly to the 12 subscribers as shown in Figure 1.12.

Figure 1.12 Using propagators to replicate to many subscribers

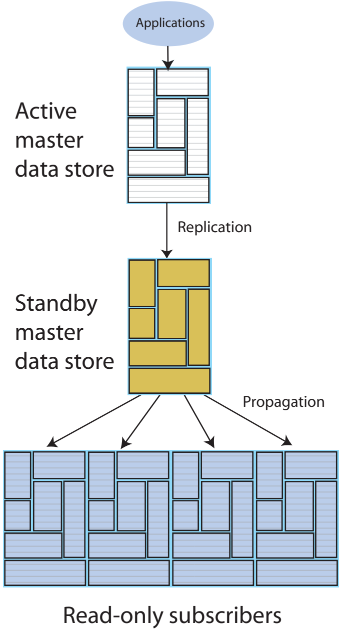


Note: Each propagator is *one-hop*, which means that you can forward an update only once. You cannot have a hierarchy of propagators where propagators forward updates to other propagators.

Active standby pair with read-only subscribers

Figure 1.13 shows an active standby pair configuration with an active master data store, a standby master data store, and four read-only subscriber data stores.

Figure 1.13 Active standby pair



Use the CREATE ACTIVE STANDBY PAIR SQL statement to create an active standby pair. The CREATE ACTIVE STANDBY PAIR statement specifies an active master data store, a standby master data store, the subscriber data stores, and the tables and cache groups that comprise the data stores.

Note: The other replication schemes in this chapter are created by using the CREATE REPLICATION statement.

In an active standby pair, two data stores are defined as masters. One is an active master data store, and the other is a standby master data store. The application updates the active master data store directly. The standby master data store cannot be updated directly. It receives the updates from the active master data store and propagates the changes to as many as 127 read-only subscriber data stores. This arrangement ensures that the standby master data store is always ahead of the subscriber data stores and enables rapid failover to the standby data store if the active master data store fails.

Only one of the master data stores can function as an active master data store at a specific time. The **ttRepStateSet** procedure assigns the role of a master data store. If the active master data store fails, then the user can use the **ttRepStateSet** procedure to change the role of the standby master data store to active before recovering the failed data store as a standby data store. The user must also start the replication agent on the new standby master data store.

If the standby master data store fails, then the active master data store can replicate changes directly to the subscribers. After the standby master data store has been recovered, it contacts the active standby data store to receive any updates that have been sent to the subscribers while the standby was down or was recovering. When the active and the standby master data stores have been synchronized, then the standby resumes propagating changes to the subscribers.

For details about setting up an active standby pair, see "Active standby pair" on page 94. For more information about administering an active standby pair, see Chapter 7, "Administering an Active Standby Pair."

Cache groups and replication

As described in the *TimesTen Cache Connect to Oracle Guide*, a *cache group* is a group of tables stored in a central Oracle database that are cached in a local TimesTen data store. This section describes how cache groups can be replicated between TimesTen data stores. The recommended way to replicate cache groups is to use an active standby pair replication scheme, due to its ability to simplify recovery in case of a failure. See "Active standby pairs with cache groups" on page 153 for more information on this method of replicating cache groups for high availability. See "Replicating cache groups" on page 73 for details on other, less recoverable, methods of replicating cache groups.

For most of the cache group replication examples in this section, assume unless otherwise indicated that there is a database server running the Oracle database and two application servers that host the TimesTen data stores, named A and B. The TimesTen data stores on A and B are caching subsets of the same tables in the Oracle database.

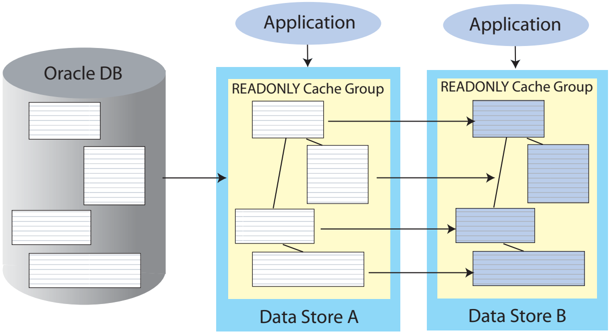
This section describes four examples of replicating cache groups:

- Replicating READONLY cache groups to regular tables
- Replicating READONLY cache groups to cache groups using active standby pair replication
- Replicating ASYNCHRONOUS WRITETHROUGH cache groups using active standby pair replication

Replicating READONLY cache groups to regular tables

You need the cache groups in data stores A and the tables in B to contain identical subsets of the same catalog information stored on Oracle. This information might change every four hours.

Figure 1.14 Replicating READONLY cache groups to regular tables



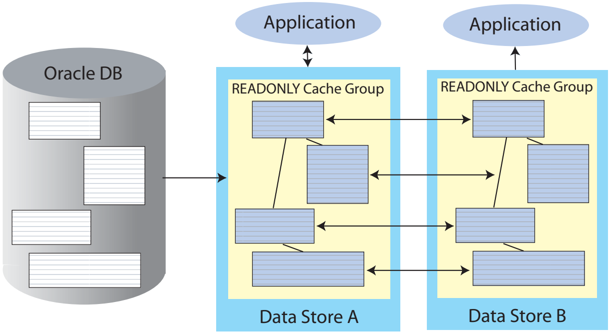
As shown in Figure 1.14, you can create a READONLY cache group in data store A and normal tables with identical column definitions in data store B. You can cache the catalog information in the cache group in data store A, and set up unidirectional replication from the cache group in A to the tables in B. You can then set the AUTOREFRESH INTERVAL value for the cache group in data store A to refresh the catalog information every four hours.

This configuration can be useful for spreading the application load across two or more data stores, but if data store A fails, recovery takes some time, as all of the data stores need to recreated to ensure that they are in sync with the catalog information in the central database.

Replicating READONLY cache groups to cache groups using active standby pair replication

You need the cache groups in data stores A and B to contain identical subsets of the same catalog information stored on Oracle. This information might change every four hours. Also, data store B needs to be able to take over for data store A in the event that data store A fails.

Figure 1.15 Replicating READONLY cache groups with active standby pair replication



As shown in Figure 1.15, you can create a READONLY cache group that, after the active standby pair rollout procedure, exists in both data stores A and B. Data store A acts as the *active master* data store in the active standby pair, and data store B acts as the *standby master.* An active standby paid allows you to cache the catalog information in the cache group in data store A, with bidirectional replication between the cache group in A and the cache group in B. You can set the AUTOREFRESH INTERVAL to refresh the catalog information every four hours, although AUTOREFRESH is automatically set to the PAUSED state on the cache group in data store B as part of the active standby pair rollout procedure, so only the cache group on data store A is updated by the AUTOREFRESH mechanism. When updates to the catalog information are autorefreshed to the cache group in data store A, replication automatically transfers the updates to the cache group in B. In addition, replication transfers the bookkeeping information necessary to allow the cache group on B to take over as the autorefreshed cache group if data store A fails.

In the event that data store A fails, data store B can quickly be reconfigured as the active master data store in the active standby pair. This automatically sets the AUTOREFRESH configuration for the cache group in data store B to the ON state, allowing data store B to take over as the autorefreshed cache group. This configuration allows quick recovery from a data store failure, with minimal interruption of data store availability.

Replicating ASYNCHRONOUS WRITETHROUGH cache groups using active standby pair replication

You need to cache multiple identical subsets of the same customer profile instance from Oracle. The profiles are occasionally updated in TimesTen, and never in Oracle, and they are read by applications connected to TimesTen very frequently. In addition, the entire configuration must be robust, with little chance of downtime.

Figure 1.16 Replicating ASYNCHRONOUS WRITETHROUGH cache groups with active standby pair replication

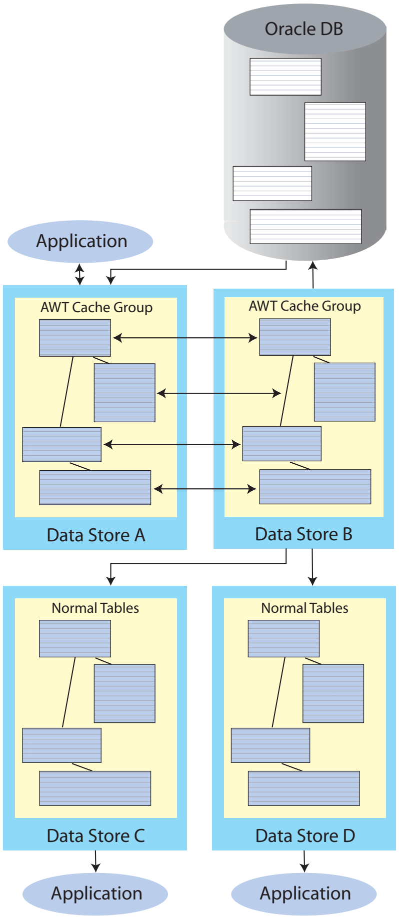


Figure 1.16 shows that you can create an ASYNCHRONOUS WRITETHROUGH cache group in data store A that, after the active standby pair rollout procedure, also exists in data store B. The cache groups in data store A and B replicate to each other and to standard tables in data stores C and D.

Application reads are distributed amongst the *read-only subscriber* data stores C and D, and application updates are made to the *active master data* store A. Updates made to data store A are automatically replicated using bidirectional replication to *standby master* data store B, which then transfers the updates to the Oracle database and replicates the updates to data stores C and D. Note that updates may not be made directly to data store B while it is the standby master.

If either the active or the standby master data store fails, the remaining master data store can take on the role of the failed data store, in addition to its normal role, until the failed data store recovers. For example, if data store A fails, data store B can accept application updates, and also continue to transfer the updates to the Oracle database as well as replicate them to data stores C and D.

If a read-only subscriber data store fails, it can be recreated from the standby master data store, with all application reads occurring from the surviving read-only subscriber data store in the meantime. For example, if data store C fails, application reads can continue to made from data store D, while data store C is recreated from data store B.

See "Active standby pairs with cache groups" on page 153 for more information.

Sequences and replication

In some replication configurations, you may find a need to keep sequences synchronized between two or more data stores. For example, you may have a master data store containing a replicated table that uses a sequence to fill in the primary key value for each row. The subscriber data store is used as a hot backup for the master data store. If updates to the sequence's current value are not replicated, insertions of new rows on the subscriber after the master has failed could conflict with rows that were originally inserted on the master.

TimesTen replication allows the incrementation of a sequence's current value to be replicated to subscriber data stores, ensuring that rows in this configuration inserted on either data store does not conflict. See "Replicating sequences" on page 81 for details on writing a replication scheme to replicate sequences.

Foreign keys and replication

If a table with a foreign key configured with ON DELETE CASCADE is replicated, then the matching foreign key on the subscriber must also be configured with ON DELETE CASCADE. In addition, you must replicate any other table with a foreign key relationship to that table. This requirement prevents foreign key conflicts from occurring on subscriber tables when a cascade deletion occurs on the master data store.

TimesTen replicates a cascade deletion as a single operation, rather than replicating to the subscriber each individual row deletion which occurs on the child table when a row is deleted on the parent. As a result, any row on the child table on the subscriber data store, which contains the foreign key value that was deleted on the parent table, is also deleted, even if that row did not exist on the child table on the master data store.

Table and cache group aging and replication

When a table or cache group is configured with either Least-Recently Used (LRU) or time-based aging, the following rules apply to the interaction with replication:

- If the replication scheme is an active standby pair, then aging is only performed on the active master data store, and deletes performed by aging are replicated to the standby master data store. Also, aging is automatically turned on on the standby master in the event that it takes over for the active master after a data store failure.
- In any other replication configuration, aging is performed individually on each data store, and deletes performed by aging are not replicated to other data stores. If LRU aging is being used, this means that the data available in the cache group on each data store may differ depending on when the data has last been accessed by applications.

Note: The aging configuration on replicated tables and cache groups must be identical on every peer data store. This is true even for tables and cache groups that are replicated as part of an active standby pair. Although table updates caused by aging are replicated to the standby data store in an active standby pair, the aging configuration must be set to ON on both the active and standby data stores. TimesTen automatically determines which data store is actually performing the aging based on its current role as active or standby.

Note: Delete operations on a cache group caused by aging are not propagated to the Oracle database when using Asynchronous or Synchronous Writethrough (AWT or SWT) cache groups.

Replication schemes

A *replication scheme* defines the configuration of data stores and replicated elements for a particular deployment of TimesTen. Replication schemes are created, altered, and deleted with SQL statements.

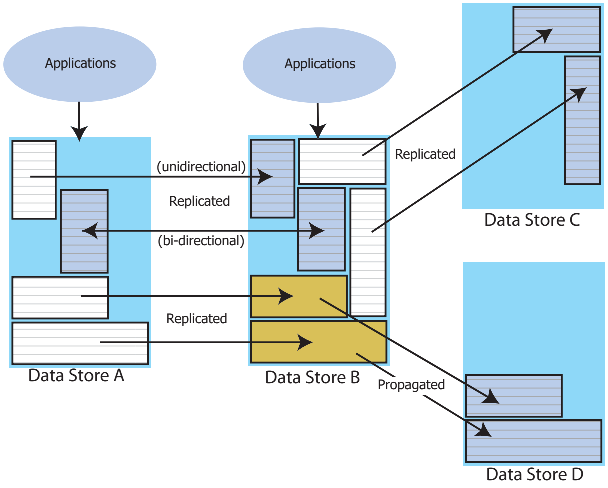
A replication scheme begins with a CREATE REPLICATION statement and assigns:

- The elements that describe the data set to be replicated
- The names of the master and subscriber data stores, as well as their server locations and their relationships, as described in "Master/subscriber relationships" on page 18

A replication scheme may optionally include the names of propagator data stores and attributes to configure a return service, transmit durability, conflict resolution, port number, log threshold, timeout period, and so on.

A scheme can define multiple roles for a single data store. Consider the example shown in Figure 1.17. Here an application makes updates on selected elements in data store B, which replicates them to data store C. Data store B also propagates selected elements between data stores A and D, and is a subscriber for other elements from data store A.

Figure 1.17 Data store with multiple roles



Note: The purpose of the example shown in Figure 1.17 is to illustrate the potential multiple roles of data stores. Although you could configure data stores in this manner, the performance and recovery issues would make such a complex configuration impractical for most deployments.

2

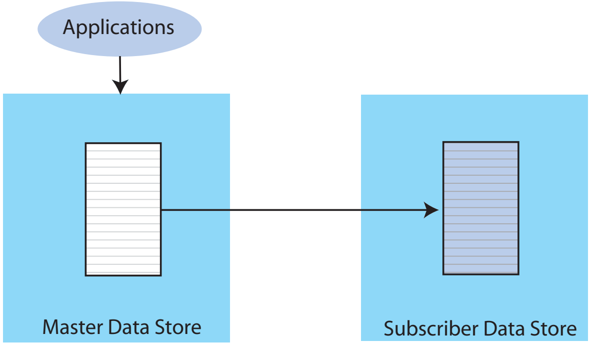
Quick Start

This chapter describes how to configure, start up, and operate a simple replication scheme. See Chapter 3, "Defining Replication Schemes" and Chapter 4, "Setting Up a Replicated System" for more details on each step.

A simple replication scheme

This section describes how to configure a simple replication scheme that replicates the contents of a single table in a master data store to a subscriber data store. To keep the example simple, both data stores reside on the same computer.

Figure 2.1 Simple Replication Scheme



The steps are:

- Step 1: Create a master and subscriber data store
- Step 2: Create a table and replication scheme
- Step 3: Start the replication agents
- Step 4: Insert data into the replicated table
- Step 5: Drop the replication scheme and table

Note: If TimesTen was installed with Access Control enabled, you must have ADMIN privileges to the data store to complete the procedures in this section. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

Step 1: Create a master and subscriber data store

Create two system data sources (System DSNs), named *masterDSN* and *subscriberDSN*, as described in Chapter 1, "Creating TimesTen Data Stores" of *Oracle TimesTen In-Memory Database Operations Guide*.

Note: Each data store "name" specified in a replication scheme must match the the prefix of the file name (without the path) given for the **DataStore** attribute in the DSN definition for the data store. A replication scheme that uses the names specified in the **Data Source Name** attributes do not work. To avoid confusion, use the same name for both your **DataStore** and **Data Source Name** attributes in each DSN definition. For example, if the data store path is `directory/subdirectory/foo.ds0`, then `foo` is the data store name that should be used in the CREATE REPLICATION statement.

Figure 2.2 Master and Subscriber Data Stores

For `masterds`, set:

- Data Store Path and Name:

```
Unix:      DataStore=/tmp/masterds
Windows:   c:\temp\masterds
```

- Permanent Data Sz (MB): **16**
- Temporary Data Sz (MB): **16**
- Database Character Set: **WE8ISO8859P1**

Use defaults for all other settings.

For `subscriberds`, set:

- Data Store Path and Name:

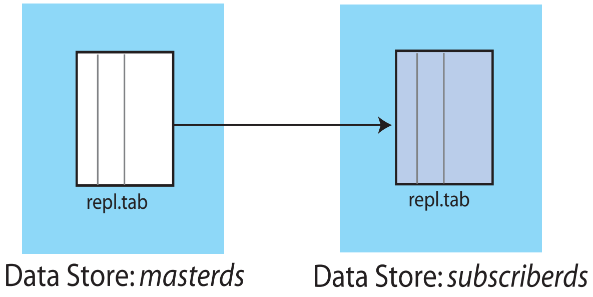
```
Unix:      DataStore=/tmp/subscriberds
Windows:   c:\temp\subscriberds
```

- Permanent Data Sz (MB): **16**
- Temporary Data Sz (MB): **16**
- Database Character Set: **WE8ISO8859P1**

Use defaults for all other settings.

Step 2: Create a table and replication scheme

Figure 2.3 Replicating `repl.tab` from master to subscriber



Use your text editor to create a SQL file, named `repscheme.sql`, and enter:

- A CREATE TABLE statement to create an empty table, named `repl.tab`, with three columns named a, b, and c
- A CREATE REPLICATION statement to define a replication scheme, named `repl.scheme`, to replicate the `repl.tab` table from the master data store to the subscriber

The contents of `repscheme.sql` should look like the following:

Example 2.1

```
CREATE TABLE repl.tab (a NUMBER NOT NULL,
                       b NUMBER,
                       c CHAR(8),
                       PRIMARY KEY (a));

CREATE REPLICATION repl.repscheme
ELEMENT e TABLE repl.tab
   MASTER masterds
   SUBSCRIBER subscriberds;
```

Open a command prompt window and use the **ttIsql** utility to apply the SQL commands specified in the `repscheme.sql` file to both the master and subscriber data stores:

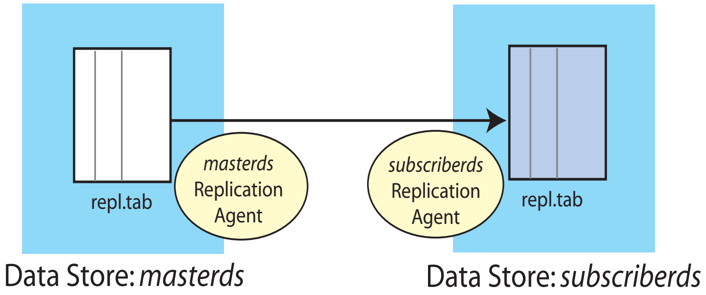
```
> ttIsql -f repscheme.sql masterds
> ttIsql -f repscheme.sql subscriberds
```

Step 3: Start the replication agents

Use the **ttAdmin** utility to start the master and subscriber replication agents:

```
> ttAdmin -repStart masterds
> ttAdmin -repStart subscriberds
```

Figure 2.4 Master and Subscriber Replication Agents



The output for each **ttAdmin** `-repStart` command should show 'Replication Manually Started: True'.

Example 2.2

```
> ttAdmin -repStart masterds
RAM Residence Policy            : inUse
Manually Loaded In Ram          : False
Replication Agent Policy        : manual
Replication Manually Started    : True
Oracle Agent Policy             : manual
Oracle Agent Manually Started   : False

> ttAdmin -repStart subscriberds
RAM Residence Policy            : inUse
Manually Loaded In Ram          : False
Replication Agent Policy        : manual
Replication Manually Started    : True
Oracle Agent Policy             : manual
Oracle Agent Manually Started   : False
```

Step 4: Insert data into the replicated table

In the command prompt window, use **ttIsql** to connect to the master data store, and INSERT some rows into the repl.tab table:

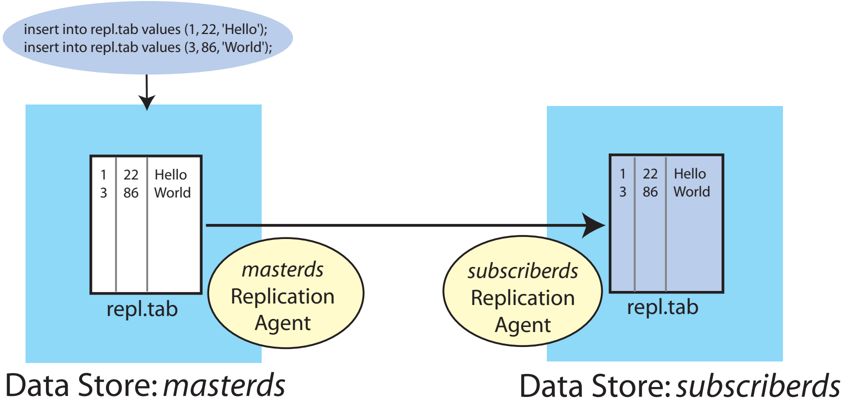
```
> ttIsql masterds
Command> INSERT INTO repl.tab VALUES (1, 22, 'Hello');
Command> INSERT INTO repl.tab VALUES (3, 86, 'World');
```

Open a second command prompt window for the subscriber, connect to the subscriber data store, and check the contents of the repl.tab table:

```
> ttIsql subscriberds
Command> SELECT * FROM repl.tab;
< 1, 22, Hello>
< 3, 86, World>
2 rows found.
```

Note: Under some circumstances, there may be a short delay before the data is available on the subscriber.

Figure 2.5 Replicating Changes to Subscriber Data Store


insert into repl.tab values (1, 22, 'Hello');
insert into repl.tab values (3, 86, 'World');
1 22 Hello
3 86 World
repl.tab
masterds Replication Agent
subscriberds Replication Agent
1 22 Hello
3 86 World
repl.tab
Data Store: masterds
Data Store: subscriberds

Any further changes you make to the repl.tab table in the masterds data store is replicated on the table in the subscriberds data store.

If you were able to replicate from masterds to subscriberds, continue to Step 5: Drop the replication scheme and table. Otherwise, review the troubleshooting tips in "Problems replicating?" on page 38.

Step 5: Drop the replication scheme and table

After you have completed your replication tests, exit **ttIsql** and use the **ttAdmin** utility to stop the master and subscriber replication agents:

```
Command> exit
> ttAdmin -repStop masterds
> ttAdmin -repStop subscriberds
```

To remove the `repl.tab` table and `repl.scheme` replication scheme from the master and subscriber data stores, use your text editor to create another SQL file, called `dropRepscheme.sql`, with the contents:

Example 2.3

```
DROP REPLICATION repl.scheme;
DROP TABLE repl.tab;
```

Note: You must drop the replication scheme *before* dropping a replicated table. Otherwise, you receive a “Cannot drop replicated table or index” error.

In a command prompt window, use the **ttIsql** utility to apply the SQL commands specified in the `dropRepscheme.sql` file to both the master and subscriber data stores:

```
> ttIsql -f dropRepscheme.sql masterds
> ttIsql -f dropRepscheme.sql subscriberds
```

Problems replicating?

If, after modifying the `repl.tab` table for `masterds` as described in Step 4: Insert data into the replicated table, the `repl.tab` table is empty on `subscriberds`:

```
> ttIsql subscriberds
Command> SELECT * FROM repl.tab;
0 rows found.
```

then there is something wrong with the replication between the `masterds` and `subscriberds` data stores.

For troubleshooting information, see "Troubleshooting Replication" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

3

Defining Replication Schemes

This chapter describes how to design a highly available system and define replication schemes. It includes the following topics:

- Designing a highly available system
- Defining a replication scheme
- Using a return service
- Creating multiple replication schemes
- Replicating materialized views
- Replicating cache groups
- Replicating sequences
- Example replication schemes
- Creating replication schemes with scripts

Designing a highly available system

As described in Chapter 1, the primary objectives of any replication scheme are to:

- Provide one or more backup data stores to ensure the data is always available to applications.
- Provide a means to recover failed data stores from their backup stores.
- Efficiently distribute workloads to provide applications with the quickest possible access to the data.
- Enable software upgrades and maintenance without disrupting service to users.

Physical configuration of hosts

When designing a highly available system, the subscriber data store must be able to survive failures that may affect the master. At a minimum, the master and subscriber need to be on separate machines. For some applications, you may want to place the subscriber in an environment that has a separate power supply. In certain cases, you may need to place a subscriber at an entirely separate site.

Efficiency and economy

Configure your data stores to best distribute application workloads and make the best use of a limited number of server machines. For example, it might be more efficient and economical to configure your data stores bidirectionally in a distributed workload manner so that each serves as both master and subscriber, rather than as separate master and subscriber data stores in a “hot standby” configuration. However, a distributed workload scheme works best with applications that primarily read from the data stores. Implementing a distributed workload scheme for applications that frequently write to the same elements in a data store may diminish performance and require that you implement a solution to prevent or manage update conflicts, as described in “Replication conflict detection and resolution” on page 167.

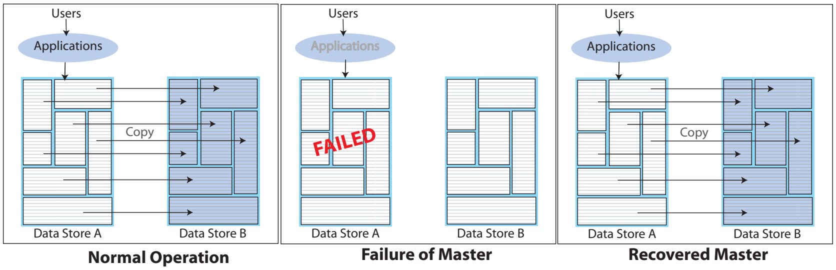
Failover and recovery

As you plan your replication scheme, consider every conceivable failover and recovery scenario. For example, subscriber failures generally have no impact on the applications connected to the master data stores and can be recovered from without disrupting user service. On the other hand, should a failure occur on a master data store, you should have a means to redirect the application load to a subscriber and continue service with no or minimal interruption. This process is typically handled by a “cluster manager” or custom software designed to detect failures, redirect users or applications from the failed data store to one of its subscribers, and manage recovery of the failed data store.

When planning your failover strategies, consider which subscriber(s) are to take on the role of its master and for which users or applications. Also consider recovery factors. For example, a failed master must be able to recover its data store from its most up-to-date subscriber, and any subscriber must be able to recover from its master.

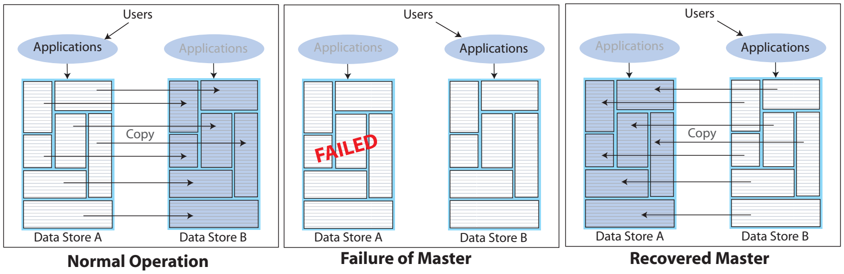
Consider the failure scenario for unidirectionally replicated data stores shown in Figure 3.1. In the case of a master failure, the application cannot access the data store until it is recovered from the subscriber. There is no way to switch the application connection or user load to the subscriber, unless you use an ALTER REPLICATION statement to redefine the subscriber data store as the master.

Figure 3.1 Recovering a master in a unidirectional scheme



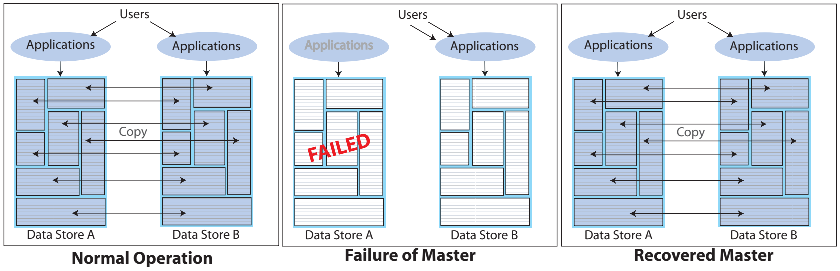
Failover and recovery are more efficient when the data stores are configured in a bidirectional general-workload scheme, such as the hot standby scheme shown in Figure 3.2. In the hot-standby scheme, should the master data store fail, the cluster manager need only shift the user load to the “hot standby” application on the subscriber data store. Upon recovering the failed data store, you can resume replication with the master/subscriber roles reversed with minimal interruption to service.

Figure 3.2 Recovering a master in a hot standby scheme



The failover procedure for data stores configured using a distributed workload scheme, such as the one shown in Figure 3.3, is similar to that used for the hot standby, only failover involves shifting the users affected by the failed data store to join the other users of an application on a surviving data store. Upon recovery, the workload can be redistributed to the application on the recovered data store.

Figure 3.3 Recovering a master in a distributed workload scheme



Performance and recovery trade-offs

When designing your replication scheme, you should weigh operational efficiencies against the complexities of failover and recovery. Factors that may complicate failover and recovery include the network topology that connects a master with its subscribers and the complexity of your replication scheme. For example, it is easier to recover a master that has been fully replicated to a single subscriber than recover a master that has selected elements replicated to different subscribers.

As described in "How replication works" on page 11, you can configure replication to work either asynchronously, "semi-synchronously" with the return receipt service, or fully synchronously with the return twosafe service. The following sections summarize the behavior of the asynchronous, return receipt, and return twosafe modes and contrast how these behaviors impact replication performance and your ability to recover from a failure. The discussions assume two data stores configured in a bidirectional, hot standby replication scheme, and replication of the entire data store.

- Commit sequence
- Performance on master
- Effect of a runtime error
- Failover (after master failure)
- Impact of TRANSMIT DURABLE/NONDURABLE on master data store recovery
- Recovery of a subscriber data store

For more information on failover and recovery, see "Managing data store failover and recovery" on page 183.

Commit sequence

- **Asynchronous** and **Return Receipt**: Each transaction is committed first on the master data store.
- **Return Twosafe**: Each transaction is committed first on the subscriber data store.

Performance on master

- **Asynchronous**: Shortest response time and best throughput because there is no long wait between transactions or before the commit on the master.
- **Return Receipt**: Longer response time and less throughput than asynchronous. The longer response time is due to the application being blocked for the duration of the network round-trip after commit. Though there is no wait before the commit on the master, replicated transactions are more serialized than with asynchronous replication, which results in less throughput.

- **Return Twosafe**: Longest response-time and least throughput. The longer response time is due to the application being blocked for the duration of the network round-trip and remote commit on the subscriber before the commit on the master. Because the commit must occur on the subscriber before the master, transactions are fully serialized, which results in the least throughput of the three modes.

Effect of a runtime error

- **Asynchronous** and **Return Receipt**: Because the transaction is first committed on the master data store, errors that occur when committing on a subscriber require the subscriber to be either:
  - manually corrected
  - destroyed and then recovered from the master data store
- **Return Twosafe**: Because the transaction is first committed on the subscriber data store, errors that occur when committing on the master require the master to be either:
  - manually corrected
  - destroyed and then recovered from the subscriber data store

Note: In twosafe mode, it is an error if a commit to succeeds on the subscriber and fails on the master. In this event, the error is likely to be fatal, requiring the master to be destroyed and then recovered from the subscriber data store.

Failover (after master failure)

- **Asynchronous** and **Return Receipt**: If the master fails and the subscriber takes over, the subscriber may be behind the master and so must reprocess data feeds and be able to remove duplicates.
- **Return Twosafe**: If the master fails and the subscriber takes over, the subscriber is at least be up-to-date with the master. It is also possible for the subscriber to be "ahead" of the master, should the master fail before committing a transaction it had replicated to the subscriber.

Impact of TRANSMIT DURABLE/NONDURABLE on master data store recovery

As described in "How replication agents copy updates between data stores" on page 12 and "Setting transmit durability on data store elements" on page 49, a master data store can be either "durable" or "non-durable." Master data stores configured for asynchronous or return receipt replication are durable by default but can be set to nondurable using the TRANSMIT NONDURABLE option in the CREATE REPLICATION statement. Master data stores configured for return twosafe replication are nondurable by default and *cannot* be made durable.

In general, if a master data store fails, you have to initiate the **ttRepAdmin** `-duplicate` operation described in "Recovering a failed data store" on page 187 to recover the failed master from the subscriber data store. This is always true for a master data store configured with TRANSMIT DURABLE.

A data store configured with TRANSMIT NONDURABLE is recovered automatically by the subscriber replication agent if it is configured in the specific type of hot-standby scheme described in "Automatic catch-up of a failed master data store" on page 185. Otherwise, you must follow the procedures described in "Recovering NONDURABLE data stores" on page 189 to recover a failed nondurable data store.

Recovery of a subscriber data store

If a subscriber in any type of replication configuration fails, you can recover it and restart replication. Some transaction records may be missing on the recovered subscriber, but the master re-sends all of the records associated with unacknowledged transactions to the subscriber. (This is the acknowledgement the subscriber replication agent sends to the master replication agent, not the application-level acknowledgements enabled by the return services.) The subscriber automatically removes any duplicate records.

Alternatively, you can initiate the **ttRepAdmin** `-duplicate` operation described in "Recovering a failed data store" on page 187 to recover a failed subscriber.

See "Subscriber failures" on page 184 for details.

Defining a replication scheme

After you have designed your replication scheme, as described in "Designing a highly available system" on page 39, you can use the CREATE REPLICATION SQL statement to apply the scheme to your data stores.

Note: To create an active standby pair, you must use the CREATE ACTIVE STANDBY PAIR SQL statement. See "Active standby pair" on page 94.

Note: If TimesTen was installed with Access Control enabled, you must have DDL privileges to the data store to use CREATE REPLICATION statement. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

The complete syntax for the CREATE REPLICATION statement is provided in the *Oracle TimesTen In-Memory Database SQL Reference Guide*. Table 3.1 shows the components of a simple replication scheme and identifies the parameters associated with the topics in this section.

Table 3.1 Components of a Replication Scheme

Component	See...
CREATE REPLICATION *Owner.SchemeName*	“Owner of the replication scheme and tables” on page 45
ELEMENT *ElementName ElementType*	“Defining replication elements” on page 46
MASTER *DataStoreName* ON "*HostName*"	
SUBSCRIBER *DataStoreName* ON "*HostName*"	
ReturnServiceAttribute	“Setting a return service attribute on table or data store elements” on page 49
STORE *DataStoreName DataStoreAttributes*;	“Setting STORE attributes” on page 52

See “Replicating sequences” on page 81 for an extensive range of sample replication schemes.

Note: Naming errors in your CREATE REPLICATION statement are often hard to troubleshoot, so take the time to check and double-check your element, data store, and host names for typos.

The replication scheme used by a data store is represented in its TTREP tables and persists across system reboots. See Chapter 7, “System and Replication Tables in the *Oracle TimesTen In-Memory Database SQL Reference Guide* for descriptions of the TTREP tables.

Note: You cannot directly modify the contents of the TTREP tables. Modifications can only be done by means of the CREATE REPLICATION or ALTER REPLICATION SQL statements.

Owner of the replication scheme and tables

The owner and name of the replication scheme and the replicated tables must be identical on both the master and subscriber data stores. To ensure you have a common owner across all data stores, you can explicitly specify an owner name with your replication scheme name in the CREATE REPLICATION statement.

For example, to assign an owner named `repl` to the replication scheme named `repscheme`, the first line of your CREATE REPLICATION statement would look like:

```
CREATE REPLICATION repl.repscheme
```

If you omit the owner from the name of your replication scheme and the replicated tables, the default owner name, as specified by the login name of the requester or the name set by the **UID** attribute in the DSN, is used in its place. Your replication scheme does not work if owner names are different across its data stores.

Defining replication elements

A replication scheme consists of one or more ELEMENT descriptions that contain the name of the element, its type (DATASTORE, TABLE or SEQUENCE), the master data store on which it is updated, and the subscriber stores to which the updates are replicated.

The name of each element in your scheme can be used to identify the element if you decide later to drop or modify the element or any of its parameters by using the ALTER REPLICATION statement. Element names must be unique within a replication scheme.

Do *not*:

- Include a specific object (table, sequence or data store) in more than one element description
- Define the same element in the role of both master and propagator
- Define an element that does not include the data store on the current host as either the master, subscriber or propagator

The correct way to define elements in a multiple subscriber scheme is described in "Multiple subscriber schemes" on page 84. The correct way to propagate elements is described in "Propagation scheme" on page 88.

You can add tables, cache groups, sequences, and data stores to an existing replication scheme by using the ALTER REPLICATION statement. See "Altering a replication scheme" on page 141. You can also drop a table or sequence from a data store that is part of an existing replication scheme. See "Dropping a table or sequence from a replication scheme" on page 145.

The rest of this section includes the following topics:

- Defining data store elements
- Defining table elements
- Defining sequence elements

Defining data store elements

To replicate the entire contents (all of the tables and sequences) of the master data store (`masterds`) to the subscriber data store (`subscriberds`), the ELEMENT description (named `ds1`) might look like the following:

```
ELEMENT ds1 DATASTORE
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
```

You can choose to exclude certain tables, sequences and cache groups from the data store element by using the EXCLUDE TABLE, EXCLUDE SEQUENCE and EXCLUDE CACHE GROUP clauses of the CREATE REPLICATION statement. When you use the EXCLUDE clauses, the entire data store is replicated to all subscribers in the element *except* for the objects that are specified in the EXCLUDE clauses. Use only one EXCLUDE clause for each kind of object (table, sequence or cache group). For example:

```
ELEMENT ds1 DATASTORE
     MASTER masterds ON "system1"
     SUBSCRIBER subscriberds ON "system2"
     EXCLUDE TABLE tab1, tab2
     EXCLUDE SEQUENCE seq1
     EXCLUDE CACHE GROUP cg3
```

You can choose to include only certain tables, sequences and cache groups in the data store by using the INCLUDE TABLE, INCLUDE SEQUENCE and INCLUDE CACHE GROUP clauses of the CREATE REPLICATION statement. When you use the INCLUDE clauses, *only* the objects that are specified in the INCLUDE clauses are replicated to each subscriber in the element. Use only one INCLUDE clause for each kind of object (table, sequence or cache group). For example:

```
ELEMENT ds1 DATASTORE
     MASTER masterds ON "system1"
     SUBSCRIBER subscriberds ON "system2"
     INCLUDE TABLE tab3
     INCLUDE SEQUENCE seq2, seq3
     INCLUDE CACHE GROUP cg1, cg2
```

When you create a new table or sequence in a data store that is configured to replicate using a data store element, the table or sequence is not replicated unless you use the ALTER REPLICATION statement to modify the replication scheme. See “Adding a table or sequence to an existing replication scheme” on page 143 for details.

Defining table elements

To replicate the `repl.tab1` and `repl.tab2` tables from a master data store (named `masterds` and located on a host named `system1`) to a subscriber data store (named `subscriberds` on a host named `system2`), your ELEMENT descriptions (named `a` and `b`) might look like the following:

```
ELEMENT a TABLE repl.tab1
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
ELEMENT b TABLE repl.tab2
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
```

Defining sequence elements

To replicate updates to the current value of the `repl.seq` sequence from a master data store (named `masterds` and located on a host named `system1`) to a subscriber data store (named `subscriberds` on a host named `system2`), your ELEMENT description (named `a`) might look like the following:

```
ELEMENT a SEQUENCE repl.seq
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
```

Note: Each data store “name” specified in a replication scheme must match the the prefix of the file name (without the path) given for the **DataStore** attribute in the DSN definition for the data store. A replication scheme that uses the names specified in the **Data Source Name** attributes does not work. To avoid confusion, use the same name for both your **DataStore** and **Data Source Name** attributes in each DSN definition. For example, if the data store path is `directory/subdirectory/foo.ds0`, then `foo` is the data store name that you should use in the CREATE REPLICATION statement.

You should identify a data store host using the host name returned by the **hostname** operating system command. If you need replication to use a different network interface than the default, you may specify one or more replication routes using the ROUTE store attribute. See “Replication with multiple network interfaces” on page 55.

Note: Host names containing special characters must be surrounded by double quotes (").

Setting additional parameters for replication elements

A data store or table element description can include additional parameters. The following sections describe them:

- Checking for replication conflicts on table elements
- Setting transmit durability on data store elements
- Setting a return service attribute on table or data store elements

Checking for replication conflicts on table elements

When data stores are configured for bidirectional replication, there is a potential for replication conflicts to occur if the same table row in two or more data stores is independently updated at the same time.

Such conflicts can be detected and resolved on a table-by-table basis by including timestamps in your replicated tables and configuring your replication scheme with the optional CHECK CONFLICTS clause in each table’s ELEMENT description.

Note: A CHECK CONFLICTS clause cannot be specified for DATASTORE elements.

See "Replication conflict detection and resolution" on page 167 for a complete discussion on replication conflicts and how to configure the CHECK CONFLICTS clause in your CREATE REPLICATION statement.

Setting transmit durability on data store elements

As described in "How replication works" on page 11, transaction records in the master data store log buffer are, by default, flushed to disk before they are forwarded to subscribers. If the entire master data store is replicated (ELEMENT is of type DATASTORE), you can improve replication performance by eliminating the master's flush-log-to-disk operation from the replication cycle. This is done by including a TRANSMIT NONDURABLE option in the ELEMENT description.

Note: When using the return twosafe service, replication is TRANSMIT NONDURABLE. Setting TRANSMIT DURABLE has no effect on return twosafe transactions.

Note: TRANSMIT DURABLE cannot be set for active standby pairs.

Example 3.1 To replicate the entire contents of the master data store (`masterds`) to the subscriber data store (`subscriberds`) and to eliminate the flush-log-to-disk operation, your ELEMENT description (named `a`) might look like:

```
ELEMENT a DATASTORE
       MASTER masterds ON "system1"
       TRANSMIT NONDURABLE
       SUBSCRIBER subscriberds ON "system2"
```

If TRANSMIT NONDURABLE is set and the master data store fails, you cannot recover the master from its log files. In this situation, follow the procedure described in "Recovering NONDURABLE data stores" on page 189.

Note: Regardless of the TRANSMIT setting, the flush-log-to-disk operation still takes place on the subscriber. See "How replication works" on page 11 for a complete description.

Setting a return service attribute on table or data store elements

As described in "How replication agents copy updates between data stores" on page 12, you can use a return receipt or return twosafe service to ensure a higher level of confidence that your replicated data is consistent on both the master and subscriber data stores.

You can specify a return service attribute independently for any subscriber defined in a CREATE REPLICATION or ALTER REPLICATION statement. Alternatively, you can specify the same return service attribute for all of the subscribers defined in an element. Example 3.2 shows separate SUBSCRIBER clauses that may define different return service attributes for *SubDataStore1* and *SubDataStore2*. Example 3.3 shows the use of a single SUBSCRIBER clause that defines the same return service attributes for both *SubDataStore1* and *SubDataStore2*.

Note: Return service attributes cannot be specified for sequence elements.

Example 3.2

```
CREATE REPLICATION Owner.SchemeName
 ELEMENT ElementName ElementType
  MASTER DataStoreName ON "HostName"
  SUBSCRIBER SubDataStore1 ON "HostName" ReturnServiceAttribute1
  SUBSCRIBER SubDataStore2 ON "HostName" ReturnServiceAttribute2;
```

Example 3.3

```
CREATE REPLICATION Owner.SchemeName
   ELEMENT ElementName ElementType
      MASTER DataStoreName ON "HostName"
      SUBSCRIBER SubDataStore1 ON "HostName",
                 SubDataStore2 ON "HostName"
                 ReturnServiceAttribute;
```

The following return service attributes can be defined for each SUBSCRIBER in your replication scheme. The use of each of these attributes is discussed in "Using a return service" on page 57.

Return Service Attribute	Description
RETURN RECEIPT	Enable the return receipt service for all transaction updates to this subscriber. See "RETURN RECEIPT" on page 58 for details.
RETURN RECEIPT BY REQUEST	Enable the return receipt service for specific transaction updates to this subscriber. See "RETURN RECEIPT BY REQUEST" on page 59 for details.

Return Service Attribute	Description
RETURN TWOSAFE	Enable the return twosafe service for all transaction updates to this subscriber. See “RETURN TWOSAFE” on page 60 for details.
RETURN TWOSAFE BY REQUEST	Enable the return twosafe service for specific transaction updates to this subscriber. See “RETURN TWOSAFE BY REQUEST” on page 62 for details.
NO RETURN	Disable the return service (either return receipt or return twosafe, depending on which service is enabled) for this subscriber (default). See “NO RETURN” on page 63 for details.

Setting STORE attributes

You can use the STORE parameter in your CREATE REPLICATION or ALTER REPLICATION statement to set optional attributes shown below for one or more data stores. The first three attributes are used to set the return service failure/ recovery policies for the data store. These attributes are discussed in "Managing return service timeout errors and replication state changes" on page 66.

STORE Attribute	Description
RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED	Continue or disable the return service when replication is stopped, as described in "Establishing return service failure/ recovery policies" on page 67.
DISABLE RETURN	Set the return service failure policy as described in "Establishing return service failure/recovery policies" on page 67.
RESUME RETURN	If return service blocking has been disabled by DISABLE RETURN, this attribute sets the policy on when to re-enable the return service.
DURABLE COMMIT	Set to override the **DurableCommits** setting on a data store and enable durable commit when return service blocking has been disabled by DISABLE RETURN.
COMPRESS TRAFFIC	Compress replicated traffic to reduce the amount of network bandwidth used. See "Compressing replicated traffic" on page 54 for details.
PORT	Set the port number used by subscriber data stores to 'listen' for updates from a master. If no PORT attribute is specified, the TimesTen daemon dynamically selects the port. While static port assignment is allowed by TimesTen, dynamic port allocation is recommended. See "Dynamic vs. static port assignments" on page 55 for details on static port assignment.

STORE Attribute	Description
TIMEOUT	Set the maximum number of seconds the data store waits before re-sending a message to an unresponsive subscriber.
RETURN WAIT TIME	Specifies the number of seconds to wait for return service acknowledgement. The default value is 10 seconds. Your application can override this timeout setting by using the *returnWait* parameter in the **ttRepSyncSet** procedure. See “Setting the return service timeout period” on page 63 for details.
LOCAL COMMIT ACTION	Specify the default action to be taken for a return service transaction in the event of a timeout. The options are: **NO ACTION** — On timeout, the commit function returns to the application, leaving the transaction in the same state it was in when it entered the commit call, with the exception that the application is not able to update any replicated tables. The application can reissue the commit. **COMMIT —** On timeout, the commit function writes a COMMIT log record and effectively ends the transaction locally. No more operations are possible on the same transaction. This default setting can be overridden for specific transactions by using the *localAction* parameter in the **ttRepSyncSet** procedure.
FAILTHRESHOLD	Set the log threshold, as described in “Setting the log failure threshold” on page 108.
CONFLICT REPORTING {SUSPEND|RESUME} AT	Specify the number of replication conflicts per second at which conflict reporting is suspended, and the number of conflicts per second at which conflict reporting resumes.

The FAILTHRESHOLD and TIMEOUT attributes can be unique to a specific replication scheme definition. This means these attribute settings can vary if you have applied different replication scheme definitions to your replicated data stores. This is *not* true for any of the other attributes, which must be the same across all replication scheme definitions. For example, setting the PORT attribute for one scheme sets it for all schemes.

For an example replication scheme that uses a STORE clause to set the FAILTHRESHOLD attribute, see Example 3.23 on page 85. For example replication schemes that set the DISABLE RETURN attribute, see Example 3.17 on page 69 and Example 3.18 on page 70.

Note: If you use CREATE REPLICATION to establish different schemes on the same data store with different PORT attributes, TimesTen ignores the setting from the last CREATE REPLICATION statement. In this case, you must use ALTER REPLICATION to change the PORT setting.

Compressing replicated traffic

If you are replicating over a low-bandwidth network, or if you are replicating massive amounts of data, you can set the COMPRESS TRAFFIC attribute to reduce the amount of bandwidth required for replication. The COMPRESS TRAFFIC attribute compresses the replicated data *from* the data store specified by the STORE parameter in your CREATE REPLICATION or ALTER REPLICATION statement. TimesTen does not compress traffic from other data stores.

Note: Though the compression algorithm is optimized for speed, enabling the COMPRESS TRAFFIC attribute has some impact on replication throughput and latency.

Example 3.4 For example, to compress replicated traffic from data store *dsn1* and leave the replicated traffic from *dsn2* uncompressed, the CREATE REPLICATION statement looks like:

```
CREATE REPLICATION repl.repscheme
ELEMENT d1 DATASTORE
        MASTER dsn1 ON machine1
        SUBSCRIBER dsn2 ON machine2
ELEMENT d2 DATASTORE
        MASTER dsn2 ON machine2
        SUBSCRIBER dsn1 ON machine1
STORE dsn1 ON machine1 COMPRESS TRAFFIC ON;
```

Example 3.5 To compress the replicated traffic between both the *dsn1* and *dsn2* data stores, use:

```
CREATE REPLICATION repl.scheme
ELEMENT d1 DATASTORE
        MASTER dsn1 ON machine1
        SUBSCRIBER dsn2 ON machine2
ELEMENT d2 DATASTORE
        MASTER dsn2 ON machine2
        SUBSCRIBER dsn1 ON machine1
STORE dsn1 ON machine1 COMPRESS TRAFFIC ON
STORE dsn2 ON machine2 COMPRESS TRAFFIC ON;
```

Dynamic vs. static port assignments

As described in "Setting STORE attributes" on page 52, if you do not assign a PORT attribute, the TimesTen daemon dynamically selects the port. When ports are assigned dynamically in this manner for the replication agents, then the ports of the TimesTen daemons have to match as well.

When statically assigning ports, it is important to specify the full host name, DSN and PORT in the STORE attribute of the CREATE REPLICATION statement.

Example 3.6 For example:

```
CREATE REPLICATION repl.repscheme
ELEMENT el1 TABLE repl.tab
        MASTER dsn1 ON machine1
        SUBSCRIBER dsn2 ON machine2
ELEMENT el2 TABLE repl.tab
        MASTER dsn2 ON machine2
        SUBSCRIBER dsn1 ON machine1
STORE dsn1 ON machine1 PORT 16080
STORE dsn2 ON machine2 PORT 16083;
```

Replication with multiple network interfaces

If your replication host has more than one network interface, you may wish to configure replication to use an interface other than the default interface. Although you must specify the host name returned by the operating system's **hostname** command when you define a replication element, you may configure replication to send or receive traffic over a different interface using the ROUTE clause.

Example 3.7 If host machine1 is configured with a second interface accessible by the host name machine1fast, and machine2 is configured with a second interface at IP address 192.168.1.100, you may specify that the secondary interfaces are used with the replication scheme:

```
CREATE REPLICATION repl.repscheme
ELEMENT e1 TABLE repl.tab
        MASTER dsn1 ON machine1
        SUBSCRIBER dsn2 ON machine2
ELEMENT e2 TABLE repl.tab
        MASTER dsn2 ON machine2
        SUBSCRIBER dsn1 ON machine1
ROUTE MASTER dsn1 ON machine1 SUBSCRIBER dsn2 ON machine2
      MASTERIP machine1fast PRIORITY 1
      SUBSCRIBERIP "192.168.1.100" PRIORITY 1
ROUTE MASTER dsn2 ON machine2 SUBSCRIBER dsn1 ON machine1
      MASTERIP "192.168.1.100" PRIORITY 1
      SUBSCRIBERIP machine1fast PRIORITY 1;
```

Alternately, on a replication host with more than one interface, you may wish to configure replication to use one or more interfaces as backups, in case the primary interface fails or the connection from it to the receiving host is broken. You may use the ROUTE clause to specify two or more interfaces for each master or subscriber that are used by replication in order of priority.

Example 3.8

If host machine1 is configured with two network interfaces at IP addresses 192.168.1.100 and 192.168.1.101, and host machine2 is configured with two interfaces at IP addresses 192.168.1.200 and 192.168.1.201, you may specify that replication use IP addresses 192.168.1.100 and 192.168.200 to transmit and receive traffic first, and to try IP addresses 192.168.1.101 or 192.168.1.201 if the first connection fails:

```
CREATE REPLICATION repl.repscheme
ELEMENT e TABLE repl.tab
        MASTER dsn1 ON machine1
        SUBSCRIBER dsn2 ON machine2
ROUTE MASTER dsn1 ON machine1 SUBSCRIBER dsn2 ON machine2
      MASTERIP "192.168.1.100" PRIORITY 1
      MASTERIP "192.168.1.101" PRIORITY 2
      SUBSCRIBERIP "192.168.1.200" PRIORITY 1
      SUBSCRIBERIP "192.168.1.201" PRIORITY 2;
```

Note: If replication on the master host is unable to bind to the MASTERIP with the highest priority, it will try to connect using subsequent MASTERIP addresses in order of priority immediately. However, if the connection to the subscriber fails for any other reason, replication will try to connect using each of the SUBSCRIBERIP addresses in order of priority before it tries the MASTERIP address with the next highest priority.

Using a return service

As described in "How replication agents copy updates between data stores" on page 12, you can configure your replication scheme with a return service to ensure a higher level of confidence that your replicated data is consistent on both the master and subscriber data stores. This section describes how to configure and manage the return receipt and return twosafe services.

The topics in this section are:

- Establishing a return service
- Setting the return service timeout period
- Checking the status of return service transactions
- Managing return service timeout errors and replication state changes

Note: The term "return service" is used when describing either the return receipt and return twosafe service. The term "return receipt" or "return twosafe" is used when describing a specific type of return service.

Establishing a return service

You select a return service for greater confidence that your data is consistent on both the master and subscriber data stores. Your decision to use either the default asynchronous, return receipt, or return twosafe mode depends on the degree of confidence you require and the performance trade-off you are willing to make in exchange. See "Performance and recovery trade-offs" on page 42 for a complete discussion of these trade-offs. In addition to the performance and recovery trade-offs between the two return services, you should also consider the following:

- Return receipt can be used in more configurations, whereas return twosafe can only be used in a bidirectional, hot-standby configuration.
- Return twosafe allows you to specify a "local action" to be taken on the master data store in the event of a timeout or other error encountered when replicating a transaction to the subscriber data store.

Note: A transaction is classified as return receipt or return twosafe when the application updates a table that is configured for either return receipt or return twosafe. Once a transaction is classified as either return receipt or return twosafe, it remains so, even if the replication scheme is altered before the transaction completes.

The following sections describe the following return service attributes:

- RETURN RECEIPT
- RETURN RECEIPT BY REQUEST
- RETURN TWOSAFE
- RETURN TWOSAFE BY REQUEST
- NO RETURN

RETURN RECEIPT

As described in "Return receipt replication" on page 14, TimesTen provides an optional *return receipt* service to loosely couple or synchronize your application with the replication mechanism.

You can specify the RETURN RECEIPT attribute to enable the return receipt service for the subscribers listed in the SUBSCRIBER clause of an ELEMENT description. With return receipt enabled, when your application commits a transaction for an element on the master data store, the application remains blocked until the subscriber acknowledges receipt of the transaction update. If the master is replicating the element to multiple subscribers, your application remains blocked until all of the subscribers have acknowledged receipt of the transaction update.

For an example replication scheme that uses return receipt services, see Example 3.20 on page 83.

Example 3.9

To confirm that all transactions committed on the `repl.tab` table in the master store (`masterds`) are received by the subscriber (`subscriberds`), your ELEMENT description (`e`) might look like the following:

```
ELEMENT e TABLE repl.tab
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
         RETURN RECEIPT
```

If any of the subscribers are unable to acknowledge receipt of the transaction within a configurable timeout period, your application receives a *tt_ErrRepReturnFailed* (8170) warning on its commit request. See "Setting the return service timeout period" on page 63 for more information on the return service timeout period.

You can use the **ttRepXactStatus** procedure to check on the status of a return receipt transaction. See "Checking the status of return service transactions" on page 64 for details.

You can also configure the replication agent to disable the return receipt service after a specific number of timeouts. See "Managing return service timeout errors and replication state changes" on page 66 for details.

Note: The RETURN SERVICES OFF WHEN REPLICATION STOPPED setting is the default setting for the return receipt service. The return receipt service is disabled if replication is stopped. See "RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED" on page 67 for details.

RETURN RECEIPT BY REQUEST

RETURN RECEIPT enables notification of receipt for all transactions. You can use RETURN RECEIPT with the BY REQUEST option to enable receipt notification only for specific transactions identified by your application.

If you specify RETURN RECEIPT BY REQUEST for a subscriber, you must use the **ttRepSyncSet** procedure to enable the return receipt service for a transaction. The call to enable the return receipt service must be part of the transaction **(AutoCommit** must be off).

Example 3.10 To enable confirmation that specific transactions committed on the `repl.tab` table in the master store (`masterds`) are received by the subscriber (`subscriberds`), your ELEMENT description (`e`) might look like:

```
ELEMENT e TABLE repl.tab
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
         RETURN RECEIPT BY REQUEST
```

Prior to committing a transaction that requires receipt notification, we call **ttRepSyncSet** within a **SQLExecDirect** function to request the return services and to set the timeout period to 45 seconds:

```
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepSyncSet(0x01, 45, NULL)", SQL_NTS )
```

If any of the subscribers are unable to acknowledge receipt of the transaction update within a configurable timeout period, your application receives a *tt_ErrRepReturnFailed* (8170) warning on its commit request. See "Setting the return service timeout period" on page 63 for more information on the return service timeout period.

You can use **ttRepSyncGet** to check if a return service is enabled and obtain the timeout value. For example:

```
Command> CALL ttRepSyncGet();
< 01, 45, 1>
1 row found.
```

For another example replication scheme that uses RETURN RECEIPT BY REQUEST, see Example 3.25 on page 86.

RETURN TWOSAFE

As described in "Return twosafe replication" on page 16, TimesTen provides a *return twosafe* service to fully synchronize your application with the replication mechanism. The return twosafe service ensures that each replicated transaction is committed on the subscriber data store before it is committed on the master data store. If replication is unable to verify the transaction has been committed on the subscriber, it returns notification of the error. Upon receiving an error, your application can either take a unique action or fall back on preconfigured actions, depending on the type of failure.

Note: The return twosafe service is intended to be used in replication schemes where two data stores must stay synchronized. One data store has an active role, while the other data store has a standby role but must be ready to assume an active role at any moment. You can use return twosafe with an active standby pair or with a bidirectional replication scheme with exactly two data stores.

Note: The return twosafe service can not be used with replicated asynchronous writethrough cache groups, except in an active standby pair configuration. See Chapter 7, "Administering an Active Standby Pair" on page 151.

To enable the return twosafe service for the subscriber, specify the RETURN TWOSAFE attribute in the SUBSCRIBER clause in your CREATE REPLICATION or ALTER REPLICATION statement.

Example 3.11

To confirm all transactions committed on the master store (`datastoreA`) are also committed by the subscriber (`datastoreB`), your ELEMENT description (`a`) might look like the following:

```
ELEMENT a DATASTORE
      MASTER datastoreA ON "system1"
      SUBSCRIBER datastoreB ON "system2"
         RETURN TWOSAFE
```

The entire CREATE REPLICATION statement that specifies both `datastoreA` and `datastoreB` in a bidirectional, hot-standby configuration with RETURN TWOSAFE might look like the following:

```
CREATE REPLICATION repl.hotstandby
ELEMENT a DATASTORE
      MASTER datastoreA ON "system1"
      SUBSCRIBER datastoreB ON "system2"
         RETURN TWOSAFE
ELEMENT b DATASTORE
      MASTER datastoreB ON "system2"
      SUBSCRIBER datastoreA ON "system1"
         RETURN TWOSAFE;
```

When replication is configured with RETURN TWOSAFE, you must disable the **AutoCommit** connection attribute.

When your application commits a transaction on the master data store, the application remains blocked until the subscriber acknowledges it has successfully committed the transaction. Initiating identical updates or deletes on both data stores can lead to deadlocks in commits that can be resolved only by stopping the processes.

If the subscriber is unable to acknowledge commit of the transaction update within a configurable timeout period, your application receives a *tt_ErrRepReturnFailed* (8170) warning on its commit request. See "Setting the return service timeout period" on page 63 for more information on the return service timeout period.

Responding to a return twosafe failure in a bidirectional replication scheme

When using the twosafe service, you can specify how the master replication agent responds to timeout errors by setting the LOCAL COMMIT ACTION attribute in the STORE clause of your CREATE REPLICATION statement or programmatically by means of the *localAction* parameter in the **ttRepSyncSet** procedure. The possible actions upon receiving a timeout during replication of a twosafe transaction are:

- COMMIT — Upon timeout, the replication agent on the master data store commits the transaction and no more operations are allowed in the transaction.
- NO ACTION — Upon timeout, the replication agent on the master data store does not commit the transaction. The process recovery commits the transaction. This is equivalent to a forced commit.

You can also configure the replication agent to disable the return twosafe service after a specific number of timeouts. See "Managing return service timeout errors and replication state changes" on page 66 for details.

If the call returns with an error related to applying the transaction on the subscriber, such as primary key lookup failure, the application can choose to rollback the transaction.

If the call returns with a type of error not mentioned above, you can use the **ttRepXactStatus** procedure described in "Checking the status of return service transactions" on page 64 to check on the status of the transaction. Depending on the error, your application can choose to:

- Reissue the commit call — This repeats the entire return twosafe replication cycle, so that the commit call returns when the success or failure of the replicated commit on the subscriber is known or if the timeout period expires.

- Roll back the transaction — If the call returns with an error related to applying the transaction on the subscriber, such as primary key lookup failure, you can rollback the transaction on the master.

If the master data store fails, then the catch-up feature described in "Automatic catch-up of a failed master data store" on page 185 automatically restores the master from the subscriber.

Note: The RETURN SERVICES ON WHEN REPLICATION STOPPED setting is the default setting for the return twosafe service, so the return twosafe service continues to block the application if replication is stopped. See "RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED" on page 67 for details.

RETURN TWOSAFE BY REQUEST

RETURN TWOSAFE enables notification of commit on the subscriber for all transactions. You can use RETURN TWOSAFE with the BY REQUEST option to enable notification of subscriber commit only for specific transactions identified by your application.

If you specify RETURN TWOSAFE BY REQUEST for a subscriber, you must use the **ttRepSyncSet** procedure to enable the return twosafe service for a transaction. The call to enable the return twosafe service must be part of the transaction **(autocommit** must be off).

Example 3.12

To enable confirmation that specific transactions committed on the master store (`datastoreA`) are also committed by the subscriber (`datastoreB`), your ELEMENT description (a) might look like:

```
ELEMENT a DATASTORE
      MASTER datastoreA ON "system1"
      SUBSCRIBER datastoreB ON "system2"
         RETURN TWOSAFE BY REQUEST;
```

Before calling commit for a transaction that requires confirmation of commit on the subscriber, we call **ttRepSyncSet** within a **SQLExecDirect** function to request the return service, set the timeout period to 45 seconds, and specify no action (1) in the event of a timeout error:

```
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepSyncSet(0x01, 45, 1)", SQL_NTS )
```

In this example, if the subscriber is unable to acknowledge commit of the transaction within the timeout period, your application receives a *tt_ErrRepReturnFailed* (8170) warning on its commit request. Your application can then chose how to handle the timeout, in the same manner as described for "RETURN TWOSAFE" on page 60.

See “Setting the return service timeout period” on page 63 for more information on setting the return service timeout period.

You can use **ttRepSyncGet** to check if a return service is enabled and obtain the timeout value. For example:

```
Command> CALL ttRepSyncGet();
< 01, 45, 1>
1 row found.
```

NO RETURN

You can use the NO RETURN attribute to explicitly disable either the return receipt or return twosafe service, depending on which one you have enabled. NO RETURN is the default condition. This attribute is typically set in ALTER REPLICATION statements. See Example 6.14 on page 148 for an example.

Setting the return service timeout period

If your replication scheme is configured with one of the return services described in “Using a return service” on page 57, a timeout occurs if any of the subscribers are unable to send an acknowledgement back to the master within a specified timeout period.

The default return service timeout period is 10 seconds. You can specify a different return service timeout period by configuring the RETURN WAIT TIME attribute in the STORE clause of your CREATE REPLICATION or ALTER REPLICATION statement, or programmatically by calling the **ttRepSyncSet** procedure with a new *returnWait* parameter. A RETURN WAIT TIME of ‘0’ indicates ‘no timeout.’

A return service may time out because of a replication failure or because replication is so far behind that the return service transaction times out before it is replicated. However, unless there is a simultaneous replication failure, failure to obtain a return service confirmation from the subscriber does not necessarily mean the transaction has not or will not be replicated.

You can set other STORE attributes to establish policies that automatically disable return service blocking in the event of excessive timeouts and re-enable return service blocking when conditions improve. See “Managing return service timeout errors and replication state changes” on page 66 for details.

Note: Once set, the timeout period applies to all subsequent return service transactions until you either reset the timeout period or terminate the application session. The timeout setting applies to all return services for all subscribers.

Example 3.13 To set the timeout period to 30 seconds for both bidirectionally replicated data stores, datastoreA and datastoreB, in the hotstandby replication scheme, the CREATE REPLICATION statement might look like the following:

```
CREATE REPLICATION repl.hotstandby
ELEMENT a DATASTORE
      MASTER datastoreA ON "system1"
      SUBSCRIBER datastoreB ON "system2"
         RETURN TWOSAFE
ELEMENT b DATASTORE
      MASTER datastoreB ON "system2"
      SUBSCRIBER datastoreA ON "system1"
         RETURN TWOSAFE
STORE datastoreA RETURN WAIT TIME 30
STORE datastoreB RETURN WAIT TIME 30;
```

Example 3.14 To use the **ttRepSyncSet** procedure to reset the timeout period to 45 seconds, call **ttRepSyncSet** within a **SQLExecDirect** function. To avoid resetting the *requestReturn* and *localAction* values, specify NULL:

```
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepSyncSet(NULL, 45, NULL)", SQL_NTS )
```

Checking the status of return service transactions

You can check the status of the last return receipt or return twosafe transaction executed on the connection handle by calling the **ttRepXactTokenGet** and **ttRepXactStatus** procedures.

First, call **ttRepXactTokenGet** to get a unique token for the last return service transaction. If you are using return receipt, the token identifies the last return receipt transaction committed on the master data store. If you are using return twosafe, the token identifies the last twosafe transaction on the master that, in the event of a successful commit on the subscriber, is committed by the replication agent on the master. However, in the event of a timeout or other error, the twosafe transaction identified by the token *is not* committed by the replication agent on the master.

Next, pass the token returned by **ttRepXactTokenGet** to the **ttRepXactStatus** procedure to obtain the return service status. The output of the **ttRepXactStatus** procedure reports which subscriber or subscribers are configured to receive the replicated data and the current status of the transaction (not sent, received, committed) with respect to each subscriber. If the subscriber replication agent encountered a problem applying the transaction to the subscriber data store, the **ttRepXactStatus** procedure also includes the error string. If you are using return twosafe and receive a timeout or other error, you can then decide whether to unconditionally commit or retry the commit, as described in “RETURN TWOSAFE” on page 60.

Note: If **ttRepXactStatus** is called without a token from **ttRepXactTokenGet**, it returns the status of the most recent transaction on the connection which was committed with the return receipt or return twosafe replication service.

The **ttRepXactStatus** procedure returns the return service status for each subscriber as a set of rows formatted as:

subscriberName, *status*, *error*

Example 3.15

For example, you can use **ttRepXactTokenGet** and **ttRepXactStatus** in a **GetRSXactStatus** function to report the status of each subscriber in your replicated system:

```
SQLRETURN GetRSXactStatus (HDBC hdbc)
{
  SQLRETURN rc = SQL_SUCCESS;
  HSTMT hstmt = SQL_NULL_HSTMT;
  char xactId [4001] = "";
  char subscriber [62] = "";
  char state [3] = "";

  /* get the last RS xact id executed on this connection */
 SQLAllocStmt (hdbc, &hstmt);
 SQLExecDirect (hstmt, "CALL ttRepXactTokenGet ('R2')", SQL_NTS);

  /* bind the xact id result as a null terminated hex string */
  SQLBindCol (hstmt, 1, SQL_C_CHAR, (SQLPOINTER) xactId,
    sizeof (xactId), NULL);

  /* fetch the first and only row */
  rc = SQLFetch (hstmt);

  /* close the cursor */
  SQLFreeStmt (hstmt, SQL_CLOSE);

  if (rc != SQL_ERROR && rc != SQL_NO_DATA_FOUND)
  {
    /* display the xact id */
    printf ("\nRS Xact ID: 0x%s\n\n", xactId);

    /* get the status of this xact id for every subscriber */
    SQLBindParameter (hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
      SQL_VARBINARY, 0, 0,
     (SQLPOINTER) xactId, strlen (xactId), NULL);

    /* execute */
    SQLExecDirect (hstmt, "CALL ttRepXactStatus (?)", SQL_NTS);

    /* bind the result columns */
    SQLBindCol (hstmt, 1, SQL_C_CHAR, (SQLPOINTER) subscriber,
      sizeof (subscriber), NULL);

    SQLBindCol (hstmt, 2, SQL_C_CHAR, (SQLPOINTER) state,
      sizeof (state), NULL);
    /* fetch the first row */
    rc = SQLFetch (hstmt);
    while (rc != SQL_ERROR && rc != SQL_NO_DATA_FOUND)
    {
      /* report the status of this subscriber */
      printf ("\n\nSubscriber: %s", subscriber);
      printf ("\nState: %s", state);
      /* are there more rows to fetch? */
      rc = SQLFetch (hstmt);
    }
  }
  /* close the statement */
  SQLFreeStmt (hstmt, SQL_DROP);
  return rc;
}
```

Managing return service timeout errors and replication state changes

The replication state can be reset to **Stop** by a user or by the master replication agent in the event of a subscriber failure. Also, as described in "Return receipt replication" on page 14 and "Return twosafe replication" on page 16, a subscriber may be unable to acknowledge a transaction that makes use of a return service and may timeout with respect to the master. If any of the subscribers are unable to acknowledge the transaction update within the timeout period, your application receives an *errRepReturnFailed* warning on its commit request.

The default return service timeout period is 10 seconds. You can specify a different return service timeout period by means of the RETURN WAIT TIME attribute in the STORE clause of your CREATE REPLICATION or ALTER REPLICATION statement, or programmatically by calling the **ttRepSyncSet** procedure with a new *returnWait* parameter.

A return service may time out or fail because of a replication failure or because replication is so far behind that the return service transaction times out before it is replicated. However, unless there is a simultaneous replication failure, failure to obtain a return service confirmation from the subscriber does not necessarily mean the transaction has not or will not be replicated.

This section describes how to detect and respond to timeouts on return service transactions. The main topics are:

- When to manually disable return service blocking
- Establishing return service failure/recovery policies

When to manually disable return service blocking

You may want respond in some manner if replication is stopped or return service timeout failures begin to adversely impact the performance of your replicated system. Your “tolerance threshold” for return service timeouts may depend on the historical frequency of timeouts and the performance/availability equation for your particular application, both of which should be factored into your response to the problem.

When using the return receipt service, you can manually respond by using ALTER REPLICATION to make changes to the replication scheme in order to disable return receipt blocking for a particular subscriber, and possibly call the **ttDurableCommit** procedure to durably commit transactions on the master that you can no longer verify as being received by the subscriber. Should you decide to disable return receipt blocking, your decision to re-enable it depends on your confidence level that the return receipt transaction is no longer likely to timeout.

An alternative to manually responding to return service timeout failures is to establish return service failure and recovery policies in your replication scheme. These policies direct the replication agents to detect changes to the replication state and to keep track of return service timeouts and then automatically respond in some predefined manner.

Establishing return service failure/recovery policies

The following attributes in your CREATE REPLICATION or ALTER REPLICATION statement set the failure/recovery policies when using a RETURN RECEIPT or RETURN TWOSAFE service:

- RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED
- DISABLE RETURN
- RESUME RETURN
- DURABLE COMMIT

The policies set by these attributes are applicable for the life of the data store or until changed. However, the replication agent must be running to enforce these policies.

RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED

The RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED attribute determines whether a return receipt or return twosafe service continues to be enabled or is disabled when replication is stopped. “Stopped” in this context means that either the master replication agent is stopped (for example, by **ttAdmin** `-repStop` *`master`*) or the replication state of the subscriber data store is set to **Stop** or **Pause** with respect to the master data store (for example, by **ttRepAdmin** `-state stop` *`subscriber`*). A failed subscriber that has exceeded the specified FAILTHRESHOLD value is set to the **Failed** state, but is eventually set to the **Stop** state by the master replication agent.

Note: A subscriber may become unavailable for a period of time that exceeds the timeout period specified by RETURN WAIT TIME but still be considered by the master replication agent to be in the **Start** state. Failure policies related to timeouts are set by the DISABLE RETURN attribute.

RETURN SERVICES OFF WHEN REPLICATION STOPPED disables the return service when replication is stopped and is the default when using the RETURN RECEIPT service. RETURN SERVICES ON WHEN REPLICATION STOPPED allows the return service to continue to be enabled when replication is stopped and is the default when using the RETURN TWOSAFE service.

Example 3.16

You have configured your CREATE REPLICATION statement to replicate updates from the `masterds` data store to the `subscriber1` data store. Your CREATE REPLICATION statement specifies the use of RETURN RECEIPT and RETURN SERVICES ON WHEN REPLICATION STOPPED.

```
CREATE REPLICATION repl.myscheme
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber1 ON "server2"
  RETURN RECEIPT
  STORE masterds ON "server1"
     RETURN SERVICES ON WHEN REPLICATION STOPPED;
```

While the application is committing updates to the master, **ttRepAdmin** is used to set `subscriber1` to the **Stop** state:

```
ttRepAdmin -dsn masterds -receiver -name subscriber1 -state stop
```

The application continues to wait for return-receipt acknowledgements from `subscriber1` until the replication state is reset to **Start** and it receives the acknowledgment:

```
ttRepAdmin -dsn masterds -receiver -name subscriber1 -state start
```

DISABLE RETURN

When a DISABLE RETURN value is set, the data store keeps track of the number of return receipt or return twosafe transactions that have exceeded the timeout period set by RETURN WAIT TIME. Should the number of timeouts exceed the maximum value set by DISABLE RETURN, your applications revert to a default replication cycle in which they no longer wait for subscribers to acknowledge the replicated updates.

You can set DISABLE RETURN SUBSCRIBER to establish a failure policy to disable return service blocking for only those subscribers that have timed out, or DISABLE RETURN ALL to establish a policy to disable return service blocking for all subscribers. You can use the **ttRepSyncSubscriberStatus** built-in procedure or the **ttRepReturnTransitionTrap** SNMP trap to determine whether a particular subscriber has been disabled by the DISABLE RETURN failure policy.

The DISABLE RETURN failure policy is only enabled when the replication agent is running. You can cancel this failure policy by stopping the replication agent and specifying either DISABLE RETURN SUBSCRIBER or DISABLE RETURN ALL with a zero value for *NumFailures*. The count of timeouts to trigger the failure policy is reset either when you restart the replication agent, when you set the DISABLE RETURN value to 0, or when return service blocking is re-enabled by RESUME RETURN.

Note: DISABLE RETURN maintains a cumulative timeout count for each subscriber. If there are multiple subscribers and you set DISABLE RETURN SUBSCRIBER, the replication agent disables return service blocking for the first subscriber that reaches the timeout threshold. Should one of the other subscribers later reach the timeout threshold, the replication agent disables return service blocking for that subscriber also, and so on.

Example 3.17

You have configured your CREATE REPLICATION statement to replicate updates from the `masterds` data store to the data stores, `subscriber1` and `subscriber2`. Your CREATE REPLICATION statement specifies the use of RETURN RECEIPT and DISABLE RETURN SUBSCRIBER with a *NumFailures* value of 5. The RETURN WAIT TIME is set to 30 seconds.

```
CREATE REPLICATION repl.myscheme
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber1 ON "server2",
             subscriber2 ON "server3"
  RETURN RECEIPT
  STORE masterds ON "server1"
     DISABLE RETURN SUBSCRIBER 5
     RETURN WAIT TIME 30;
```

While the application is committing updates to the master, `subscriber1` experiences problems and fails to acknowledge a replicated transaction update. The application is blocked 30 seconds after which it commits its next update to the master. Over the course of the application session, this commit/timeout cycle repeats 4 more times until DISABLE RETURN disables return-receipt blocking for `subscriber1`. The application continues to wait for return-receipt acknowledgements from `subscriber2` but not from `subscriber1`.

RETURN SERVICES OFF WHEN REPLICATION STOPPED is the default setting for the return receipt service. Therefore, return receipt is disabled under *either* one of the following conditions:

- The subscriber is unable to acknowledge an update within the specified RETURN WAIT TIME, as described above.

- Replication is stopped, as described in "RETURN SERVICES { ON | OFF } WHEN REPLICATION STOPPED" on page 67.

RESUME RETURN

When we say return service blocking is "disabled," we mean that the applications on the master data store no longer block execution while waiting to receive acknowledgements from the subscribers that they received or committed the replicated updates (see Figure 1.3 on page 15 and Figure 1.4 on page 16). Note, however, that the master still listens for an acknowledgement of each batch of replicated updates from the subscribers as it would in the default replication case described in "Default replication" on page 12.

You can establish a return service recovery policy by setting the RESUME RETURN attribute and specifying a resume latency value. When this attribute is set, and return service blocking has been disabled for a subscriber, the return receipt or return twosafe service is re-enabled when the commit-to-acknowledge time for a transaction falls below the value set by RESUME RETURN. The commit-to-acknowledge time is the latency between when the application issues a commit and when the master receives acknowledgement of the update from the subscriber, as shown in Steps 2 and 5 in Figure 1.3 on page 15.

Example 3.18 For example, if return-receipt blocking has been disabled for `subscriber1` and if RESUME RETURN is set to 8 milliseconds, then return-receipt blocking is re-enabled for `subscriber1` the instant it acknowledges an update in less than 8 milliseconds from when it was committed by the application on the master.

```
CREATE REPLICATION repl.myscheme
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber1 ON "server2",
             subscriber2 ON "server3"
  RETURN RECEIPT
  STORE masterds ON "server1"
     DISABLE RETURN SUBSCRIBER 5
     RESUME RETURN 8;
```

The RESUME RETURN policy is enabled only when the replication agent is running. You can cancel a return receipt resume policy by stopping the replication agent and then using ALTER REPLICATION to set RESUME RETURN to zero.

DURABLE COMMIT

You can set the DURABLE COMMIT attribute to specify the durable commit policy for applications that have return service blocking disabled by DISABLE RETURN. When DURABLE COMMIT is set to ON, it overrides the **DurableCommits** setting on the master data store and forces durable commits for those transactions that have had return service blocking disabled.

Note: If the replication scheme is configured with RETURN SERVICES ON WHEN REPLICATION STOPPED, the replication agent must be running to enforce the DURABLE COMMIT policy.

Example 3.19 For example, you can set DURABLE COMMIT ON when establishing a DISABLE RETURN ALL policy to disable return-receipt blocking for all subscribers. If return-receipt blocking is disabled, commits are durably committed to disk to provide redundancy.

```
CREATE REPLICATION repl.myscheme
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber ON "server2",
             subscriber2 ON "server3"
  RETURN RECEIPT
  STORE masterds ON "server1"
     DISABLE RETURN ALL 5
     DURABLE COMMIT ON
     RESUME RETURN 8;
```

Note: DURABLE COMMIT is also useful if you have only one subscriber. However, if you are replicating the same data to two subscribers, as shown in Example 3.17 and Example 3.18, and you disable return service blocking to one subscriber, then you achieve better performance if you rely on the other subscriber than you would by enabling durable commits.

Creating multiple replication schemes

Though it is often valid to assign more than one replication scheme to a data store, managing your replicated system is usually much easier if you contain your replication definition in a single scheme and apply that scheme to all of your replicated data stores.

However, there may be circumstances in which you want to define different replication schemes on different data stores. For example, in a large replicated system that is distributed across multiple sites, it might be more efficient for each site to autonomously manage a separate scheme. It might also be useful to create separate schemes with different SUBSCRIBER and STORE attributes to better accommodate the characteristics of the various hosts.

Note the following restrictions when creating multiple replication schemes:

- There cannot be more than one replication scheme that describes replication from one data store to another data store. For example, you cannot have two separate replication schemes that replicate from the `masterds` data store to the `subscriberds` data store:

```
CREATE REPLICATION repl.scheme1
ELEMENT e TABLE repl.tab1
  MASTER masterds
  SUBSCRIBER subscriberds;

CREATE REPLICATION repl.repscheme2
ELEMENT e2 TABLE repl.tab2
  MASTER masterds
  SUBSCRIBER subscriberds;
```

- A table for which a data store is the master in one replication scheme cannot have the same data store as a master for the same table in another replication scheme. For example, you cannot have two replication schemes that replicate the `repl.tab1` table from the `masterds` data store to the `subscriber1ds` and `subscriber2ds` data stores:

```
CREATE REPLICATION repl.repscheme1
ELEMENT e TABLE repl.tab1
  MASTER masterds
  SUBSCRIBER subscriber1ds;

CREATE REPLICATION repl.repscheme2
ELEMENT e2 TABLE repl.tab1
  MASTER masterds
  SUBSCRIBER subscriber2ds;
```

Replicating tables with foreign key relationships

Ordinarily, you may choose to replicate all or merely a subset of tables that have foreign key relationships with one another. However, if the foreign key relationships have been configured with ON DELETE CASCADE, then you must configure replication to replicate all of the tables, either by configuring the replication scheme with a DATASTORE element that does not EXCLUDE any of the tables, or by configuring the scheme with a TABLE element for every table that is involved in the relationship.

Note: As a consequence of this requirement, it is not possible to add a table with a foreign key relationship configured with ON DELETE CASCADE to a pre-existing replication scheme using ALTER REPLICATION. Instead, you must use DROP REPLICATION to drop the replication scheme, create the new table with the foreign key relationship, and then use CREATE REPLICATION to create a new replication scheme replicating all of the related tables.

Replicating materialized views

A materialized view is a summary of data selected from one or more TimesTen tables, called *detail tables*. Though you cannot replicate materialized views directly, you can replicate their underlying detail tables in the same manner as you would replicate regular TimesTen tables.

The detail tables on the master and subscriber data stores can be referenced by materialized views. However, TimesTen replication verifies only that the replicated detail tables have the same structure on both the master and subscriber. It does not enforce that the materialized views are the same on each data store.

If you replicate an entire data store containing a materialized or non-materialized view as a DATASTORE element, only the detail tables associated with the view are replicated. The view itself is not replicated. A matching view can be defined on the subscriber data store, but is not required. If detail tables are replicated, TimesTen automatically updates the corresponding view.

Materialized views defined on replicated tables may result in replication failures or inconsistencies if the materialized view is specified so that overflow or underflow conditions occur when the materialized view is updated.

Replicating cache groups

You can replicate cache groups to cache groups or to standard TimesTen tables. If a cache group is replicated, then all of the tables in the cache group must be replicated. You may not replicate some tables in a cache group and without replicating all of them.

Note: The recommended method of replicating cache groups for high availability is by using an active standby pair replication configuration. Active standby pair replication allows for quick recovery in the event of a failure. See "Active standby pairs with cache groups" on page 153 for more information.

A cache group can be replicated by one of the following methods:

- Replicate each of the tables in the cache group
- Replicate the data store that contains the cache group

See "Cache group replication scheme" on page 91 for examples.

This section includes the following topics on using standard replication to replicate cache groups:

- Using ttRepAdmin to set up replication of cache groups
- Using CREATE CACHE GROUP to set up replication of cache groups
- Unidirectional replication of cache groups to cache groups
- Bidirectional replication of cache groups to cache groups

See “Defining data store elements” on page 46 in the section about “Defining a replication scheme” on page 44 for more information about including specific cache groups in a replication scheme.

See “Recovering a failed data store” on page 187 for information about recovering a failed data store that contains cache groups.

See “Administering an Active Standby Pair” on page 151 for more information on setting up, administering and recovering replicated cache groups using active standby pair replication. You should use active standby pair replication for replicating cache groups whenever possible, in order to simplify both the initial set up of your data stores and recovery from failures.

Using ttRepAdmin to set up replication of cache groups

You can duplicate cache groups by using the **ttRepAdmin** utility with the `-duplicate` option to duplicate a data store containing cache groups. The `-duplicate` option has two options: `-keepCG` and `-noKeepCG` The `-keepCG` option preserves the cache group definitions when the cache group is duplicated. The `-noKeepCG` option does not preserve the cache group definitions. It converts cache group tables into standard TimesTen tables during duplication. Both of these options are instrumental in rolling out an active standby pair replication configuration that includes cache groups.

Use **ttRepAdmin** `-duplicate -keepCG` to set up replication between cache groups and also for failover when a master data store with cache groups fails.

Use **ttRepAdmin** `-duplicate -noKeepCG` to set up replication between a cache group and standard TimesTen tables for load balancing.

The following sections illustrate the use of the `-keepCG` and `-noKeepCG` options of **ttRepAdmin** `-duplicate`:

- Bidirectional hot standby READONLY cache groups with AUTOREFRESH: -keepCG option
- Bidirectional hot standby WRITETHROUGH cache groups: -keepCG option
- Load-balancing AUTOREFRESH cache groups: -noKeepCG option

Bidirectional hot standby READONLY cache groups with AUTOREFRESH: -keepCG option

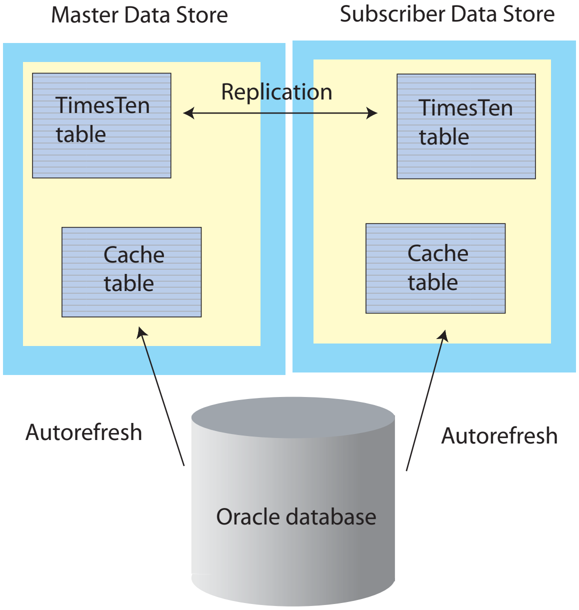
In this scenario, the master data store contains an autorefresh READONLY cache group as well as TimesTen tables that are not part of a cache group. You can use the `-keepCG` option to create a subscriber data store that preserves the autorefresh read-only cache group definition.

The cache group can be set up to be autorefreshed directly from Oracle. The cache group on the master data store receives its updates directly from Oracle, while the cache group on the subscriber automatically has its AUTOREFRESH state set to PAUSED, and receives its updates from the master. If there is a failure of the master data store, the AUTOREFRESH state on the subscriber can be set to ON and it may be used as a master data store with no data loss. The original master can then be recovered by duplicating it from the new master, or by allowing the Master Catch-Up process to automatically resynchronize them if the replication scheme has been configured with RETURN TWOSAFE.

The TimesTen tables that are not in the cache group should also be included in the bidirectional replication scheme between the master data store and the subscriber data store.

See Figure 3.4.

Figure 3.4 Bidirectional hot standby READONLY AUTOREFRESH cache groups



Complete the following tasks to set up a pair of bidirectionally replicating data stores that contain identical READONLY cache groups with AUTOREFRESH:

1. Create the master data store.
2. Set the cache agent user ID and password by calling **ttCacheUidPwdSet**. Start the cache agent for the master data store by calling **ttCacheStart** or using the **ttAdmin** `-cacheStart` command.
3. Create the READONLY cache group using the CREATE CACHE GROUP command on the master data store with the AUTOREFRESH STATE set to PAUSED (the default).

4. Create the replication scheme on the master data store using the CREATE REPLICATION statement.

5. Load the cache group on the master data store with the LOAD CACHE GROUP statement. This sets the AUTOREFRESH STATE to ON.

6. Start the replication agent for the master data store by calling **ttRepStart** or using the **ttAdmin** `-repStart` command.

7. Use the **ttRepAdmin** `-duplicate` command with the `-keepCG` option to create the subscriber data store. The `-keepCG` option sets up the autorefresh objects for the subscriber data store on the Oracle database. You must provide the cache administration user ID and password because the cache groups are autorefresh. The AUTOREFRESH STATE for the subscriber data store is automatically set to PAUSED.

8. Start the cache agent for the subscriber data store.

9. Start the replication agent for the subscriber data store.

To recover from a master data store failure, complete the following tasks:

1. Stop the cache agent on the master data store if it is not already stopped, by calling **ttCacheStop** or using the **ttAdmin** `-cacheStop` command.

2. Stop the replication agent on the master data store if it is not already stopped, by calling **ttRepStop** or using the **ttAdmin** `-repStop` command.

3. On the subscriber data store, use the ALTER CACHE GROUP statement to set the AUTOREFRESH STATE of the cache group to ON.

4. Restore the failed master data store using one of two methods:

   - If the replication scheme is configured with RETURN TWOSAFE, reconnect to the failed master data store. The Master Catch-Up process automatically synchronizes the two data stores, and the AUTOREFRESH STATE on the failed master is set to PAUSED. It now acts as the hot standby subscriber data store.
   - If the replication scheme is not configured with RETURN TWOSAFE, use the **ttRepAdmin** `-duplicate -keepCG` command to duplicate the failed master data store from the current master data store, as described in Step 7 of the instructions for setting up the two data stores.

5. Start the cache agent for the recovered data store.

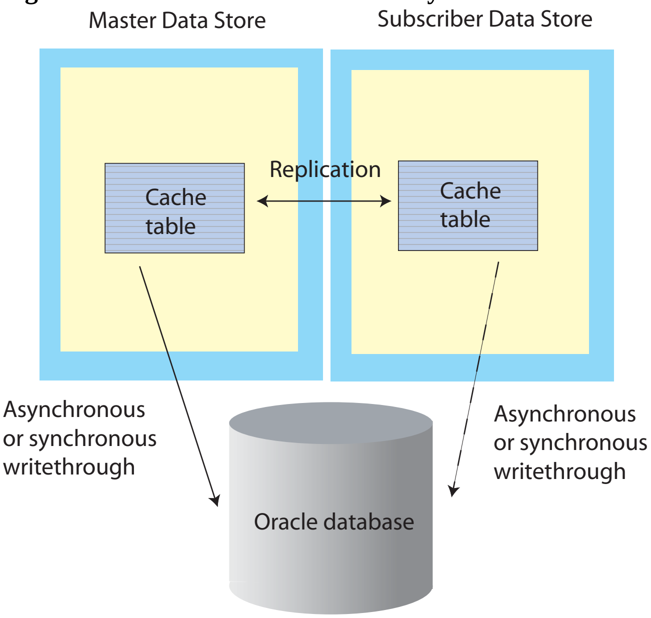
6. Start the replication agent for the recovered data store.

Note: See "Setting up an active standby pair" on page 154 for instructions on using the `-keepCG` option as part of rolling out an active standby pair with read-only AUTOREFRESH cache groups.

Bidirectional hot standby WRITETHROUGH cache groups: -keepCG option

In this scenario, the master data store contains an ASYNCHRONOUS WRITETHROUGH or SYNCHRONOUS WRITETHROUGH cache group. You can use the `-keepCG` option to create a subscriber data store that preserves the WRITETHROUGH cache group definition. See Figure 3.5.

Figure 3.5 Bidirectional hot standby WRITETHROUGH cache groups



Complete the following tasks to set up a pair of bidirectional hot standby data stores that contain identical WRITETHROUGH cache groups:

1. Create the master data store.
2. Register the cache administration user ID and password if the WRITETHROUGH cache group is ASYNCHRONOUS.
3. Create the WRITETHROUGH cache groups in the master data store and on the node on which the subscriber data store are to be created.
4. Start the replication agent for the master data store.
5. Use **ttRepAdmin** `-duplicate` with the `-keepCG` option to create the subscriber data store. The `-keepCG` option sets up the ASYNCHRONOUS WRITETHROUGH (AWT) objects for the subscriber data store on the Oracle database. You must provide the cache administration user ID and password if the cache groups are AUTOREFRESH or AWT.
6. Start the replication agent for the subscriber data store.

If the cache group is ASYNCHRONOUS WRITETHROUGH, the subscriber data store and the Oracle database may be in different replication states after a master data store failure. The application may need to replay some of the transactions to bring them into the same state.

To recover from a master data store failure, perform Steps 5 and 6 on the master data store.

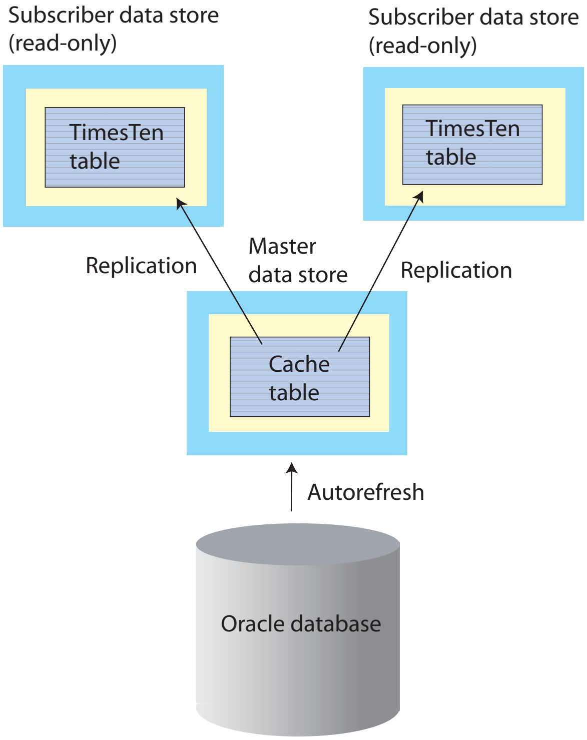
Note: The data stores can contain both AUTOREFRESH cache groups and WRITETHROUGH cache groups. The tasks in this section set up replication between the WRITETHROUGH cache groups but not between the AUTOREFRESH cache groups. See "Bidirectional hot standby READONLY cache groups with AUTOREFRESH: -keepCG option" on page 74.

Note: See "Setting up an active standby pair" on page 154 for instructions on using the `-keepCG` option as part of rolling out an active standby pair with ASYNCHRONOUS WRITETHROUGH cache groups.

Load-balancing AUTOREFRESH cache groups: -noKeepCG option

In this scenario, the master data store contains an AUTOREFRESH cache group that receives changes from the Oracle database. The master data store is replicated to subscriber data stores for read-only access. The subscriber data stores contain TimesTen tables that are not in cache groups. Autorefresh occurs only on the master data store. The subscriber data stores do not communicate with the Oracle database and you do not need to have the Oracle client installed. See Figure 3.6.

Figure 3.6 Load-balancing AUTOREFRESH cache groups



Complete the following tasks to set up replication for load-balancing AUTOREFRESH cache groups:

1. Create the master data store.
2. Start the cache agent for the master data store.
3. Create the AUTOREFRESH cache group on the master data store.
4. Create the replication scheme.
5. Start the replication agent for the master data store.
6. Use **ttRepAdmin** `-duplicate` with the `-noKeepCG` option to create the subscriber data stores.
7. Start the replication agent for the subscriber data stores.

Using CREATE CACHE GROUP to set up replication of cache groups

Another way to set up replication between cache groups is to create both cache groups with the CREATE CACHE GROUP statement and then to set up the replication scheme. Both cache groups must specify identical cache group types.

For example, a READONLY cache group can be replicated only to another READONLY cache group. In addition, any cache group attributes specified in the CREATE CACHE GROUP statement must be the same, with the exception of the AUTOREFRESH, READONLY, AGING, and PROPAGATE attributes, as described in "Unidirectional replication of cache groups to cache groups" on page 80 and "Bidirectional replication of cache groups to cache groups" on page 81.

Unidirectional replication of cache groups to cache groups

If unidirectionally replicated cache groups are created independently on each data store using the CREATE CACHE GROUP statement, there are restrictions on the valid configuration of some cache group attributes.

See "Cache group replication scheme" on page 91 for an example of a unidirectional replication scheme that replicates a cache group.

Restrictions on AUTOREFRESH configuration

If the cache group is READONLY, AUTOREFRESH is automatically set to ON when the cache group is loaded, so the AUTOREFRESH STATE on the subscriber cache group must be explicitly set to OFF or PAUSED in the CREATE CACHE GROUP statement.

If the cache group is USERMANAGED (the default) and the AUTOREFRESH STATE on the master cache group is set to ON, then the AUTOREFRESH STATE on the subscriber cache group must be set to OFF and the all of the tables in the subscriber cache group must be configured as READONLY.

If the cache group is USERMANAGED and the subscriber cache group is configured with PROPAGATE, then the cache group on both the master and subscriber may not autorefresh. The cache groups must have no AUTOREFRESH configuration, or if they do, they must both have the AUTOREFRESH STATE set to OFF.

Restrictions on AGING configuration

If time-based aging is configured, the LIFETIME setting must be identical on both the master and subscriber cache groups. Additionally, if AGING is set to ON for one cache group, it must be set to ON for both

Restrictions on the WHERE clause

The WHERE clauses must identical for the both the master and subscriber cache groups.

Bidirectional replication of cache groups to cache groups

When replicating cache groups in a bidirectional replication scheme, READONLY cache groups may specify AUTOREFRESH for both cache groups. WRITETHROUGH cache groups may only specify AUTOREFRESH for one of the cache groups. USERMANAGED cache groups may not use AUTOREFRESH.

More specifically, for:

- **READONLY cache groups -** One cache group must specify an AUTOREFRESH state of either PAUSED or OFF.
- **SYNCHRONOUS WRITETHROUGH cache groups -** No replication issues.
- **ASYNCHRONOUS WRITETHROUGH cache groups** - No replication issues
- **USERMANAGED cache groups -** Both cache groups must specify AUTOREFRESH STATE OFF.

Note: The replication agent does not recognize changes made by ALTER CACHE GROUP. If you use ALTER CACHE GROUP to reset the AUTOREFRESH STATE, you must then restart the replication agent.

Replicating sequences

You can use replication to ensure that the current value of a sequence on a subscriber data store is always in advance of the current value on the master data store, thereby preventing conflicts if the sequence is later used to make updates directly on the subscriber data store. For example, you may have an application that uses a sequence to determine primary key values in a replicated table, and a configuration that includes a hot standby data store that must assume the master role when the master data store fails. By replicating your sequence, you can guarantee that the same sequence value is not used twice, regardless of which data store you update directly.

Sequence replication works by transmitting a new current value from the master data store to the subscriber every 20 references to the sequence’s NEXTVAL, starting with the first reference. For example, consider a sequence `my.seq` with a MINVALUE of 1 and an INCREMENT of 2. The very first time that you use `my.seq.NEXTVAL` in a transaction, the current value of the sequence on the master data store is changed to three, and a new current value of 41 is replicated to the subscriber. The next 19 references to `my.seq.NEXTVAL` on the master data store result in no new current value being replicated, since the current value of 41 on the subscriber data store is still ahead of the current value on the master. Only on the twenty-first reference to `my.seq.NEXTVAL` is a new current value, 61, transmitted to the subscriber data store, as the subscriber's previous current value of 41 would now be behind the value of 43 on the master.

Sequence replication has these limitations:

- Sequences with the CYCLE attribute cannot be replicated.
- The definition of the replicated sequence on each peer data store must be identical.
- No conflict checking is performed on sequences. If you make updates to sequences in both data stores in a bidirectional replication configuration without using the RETURN TWOSAFE service, it is possible for both sequences to return the identical NEXTVAL.

If you need to use sequences in a bidirectional replication scheme where updates may occur on either peer, you may instead use a non-replicated sequence with different MINVALUE and MAXVALUE attributes on each data store. For example, you may create sequence `my.seq` on datastore DS1 with a MINVALUE of 1 and a MAXVALUE of 100, and the same sequence on DS2 with a MINVALUE of 101 and a MAXVALUE of 200. Then, if you configure DS1 and DS2 with a bidirectional replication scheme, you may make updates to either data store using the sequence `my.seq` with the guarantee that the sequence values never conflict. Be aware that if you are planning on using **ttRepAdmin** `-duplicate` to recover from a failure in this configuration, you must drop and then re-create the sequence with a new MINVALUE and MAXVALUE after you have performed the duplicate.

Note: Replicated sequences are intended to be used in conjunction with replicated tables. Therefore, sequence updates are only replicated when they are followed by or used in updates to replicated tables. Operations on sequences such as `SELECT my.seq.NEXTVAL FROM sys.dual`, while incrementing the sequence value, are not replicated until they are followed by updates to tables that are replicated. A side effect of this behavior is that these sequence updates are not purged from the log until followed by updates to tables that are replicated. This causes **ttRepSubscriberWait** and **ttRepAdmin** `-wait` to fail when only these sequence updates are present at the end of the log.

See "Defining replication elements" on page 46 for more information on configuring a replication scheme to include sequences.

Example replication schemes

The examples described in this section illustrate how to configure a variety of replication schemes. The examples have been kept simple for clarity. You can use these examples as a starting point from which to build more complex replication schemes.

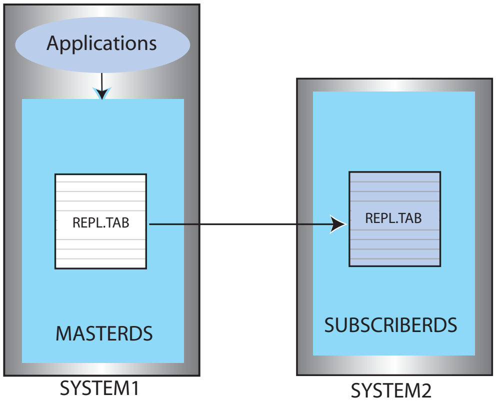
The schemes described are:

- Single subscriber scheme
- Multiple subscriber schemes
- Selective replication scheme
- Propagation scheme
- Bidirectional split workload scheme
- Bidirectional general workload scheme
- Cache group replication scheme
- Active standby pair

Single subscriber scheme

The scheme shown in Example 3.3 is based on the single master and subscriber unidirectional replication scheme described in Chapter 2, "Quick Start." However, in this example, the two data stores are located on separate hosts, `system1` and `system2`. We also make use of the RETURN RECEIPT service to confirm all transactions committed on the `repl.tab` table in the master store are received by the subscriber, as described in "Return receipt replication" on page 14.

Figure 3.7 Unidirectional replication (single table)

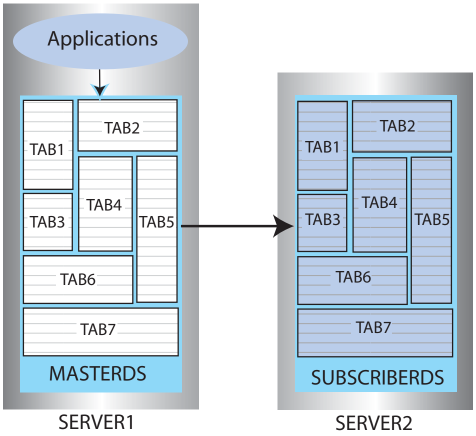


Example 3.20

```
CREATE REPLICATION repl.repscheme
  ELEMENT e TABLE repl.tab
      MASTER masterds ON "system1"
      SUBSCRIBER subscriberds ON "system2"
         RETURN RECEIPT;
```

The scheme shown in Example 3.21 establishes a master data store, named MASTERDS, that replicates its entire contents (`tab1` through `tab7`) to the subscriber data store, named `subscriberds`, located on `server2`.

Figure 3.8 Unidirectional replication (entire data store)



Example 3.21

```
CREATE REPLICATION repl.repscheme
   ELEMENT e DATASTORE
      MASTER masterds ON "server1"
      SUBSCRIBER subscriberds ON "server2";
```

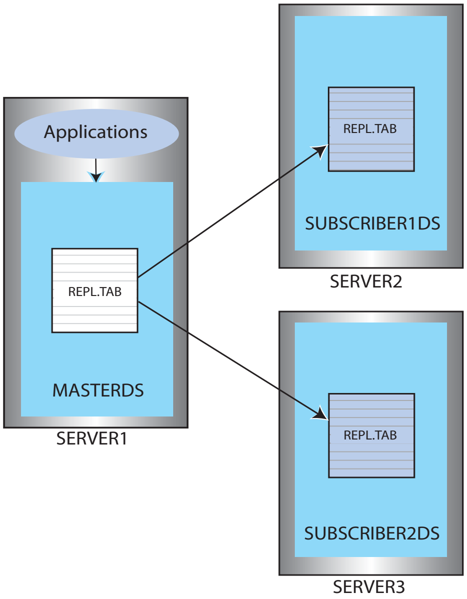
Multiple subscriber schemes

You can create a replication scheme that includes up to 128 subscriber data stores.

Figure 3.9 shows a master data store with a table (`repl.tab`) that is replicated to two subscriber data stores:

- `masterds` master data store is on `server1`
- `subscriber1ds` subscriber data store is on `server2`
- `subscriber2ds` subscriber data store is on `server3`

Figure 3.9 Replicating to multiple subscribers



Example 3.22

This example establishes a master data store, named `masterds`, that replicates the `repl.tab` table to two subscriber data stores, `subscriber1ds` and `subscriber2ds`, located on `server2` and `server3`, respectively. The name of the replication scheme is `repl.twosubscribers`. The name of the replication element is `e`.

```
CREATE REPLICATION repl.twosubscribers
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber1ds ON "server2",
             subscriber2ds ON "server3";
```

Example 3.23

This example uses the basic example in Example 3.22 and adds a RETURN RECEIPT attribute and a STORE parameter. RETURN RECEIPT enables the return receipt service for both data stores. The STORE parameter sets a FAILTHRESHOLD value of 10 to establish the maximum number of log files that can accumulate on `masterds` for a subscriber before it assumes the subscriber has failed. `CREATE REPLICATION repl.twosubscribers`

```
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber1ds ON "server2",
             subscriber2ds ON "server3"
  RETURN RECEIPT
STORE masterds FAILTHRESHOLD 10;
```

Example 3.24 This example shows how to enable RETURN RECEIPT for only `subscriber2ds` (no comma after the `subsciber1ds` definition).

```
CREATE REPLICATION repl.twosubscribers
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriber1ds ON "server2"
  SUBSCRIBER subscriber2ds ON "server3" RETURN RECEIPT
STORE masterds FAILTHRESHOLD 10;
```

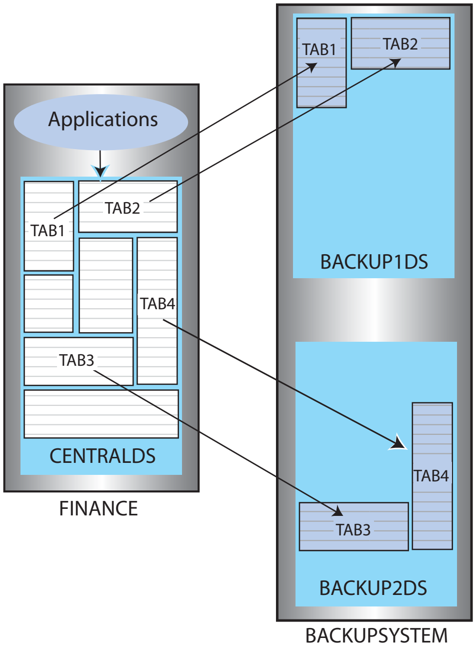
Example 3.25 This example shows how to apply RETURN RECEIPT BY REQUEST to `subscriber1ds` and RETURN RECEIPT to `subscriber2ds`. In this scheme, applications accessing `subscriber1ds` must use the **ttRepSyncSet** procedure to enable the return services for a transaction, while `subscriber2ds` unconditionally provides return services for all transactions.

```
CREATE REPLICATION repl.twosubscribers
ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER subscriberds1 ON "server2" RETURN RECEIPT BY REQUEST
  SUBSCRIBER subscriber2ds ON "server3" RETURN RECEIPT
STORE masterds FAILTHRESHOLD 10;
```

Selective replication scheme

The selective replication scheme shown in Example 3.26 establishes a master data store, named `centralds`, that replicates four tables. `tab1` and `tab2` are replicated to the subscriber `backup1ds`. `tab3` and `tab4` are replicated to `backup2ds`. The master data store is located on the `finance` server.Both subscribers are located on the `backupsystem` server.

Figure 3.10 Selective replication



Example 3.26

```
CREATE REPLICATION repl.twobackups
ELEMENT a TABLE tab1
  MASTER centralds ON "finance"
  SUBSCRIBER backup1ds ON "backupsystem"
ELEMENT b TABLE tab2
  MASTER centralds ON "finance"
  SUBSCRIBER backup1ds ON "backupsystem"
ELEMENT d TABLE tab3
  MASTER centralds ON "finance"
  SUBSCRIBER backup2ds ON "backupsystem"
ELEMENT d TABLE tab4
  MASTER centralds ON "finance"
  SUBSCRIBER backup2ds ON "backupsystem";
```

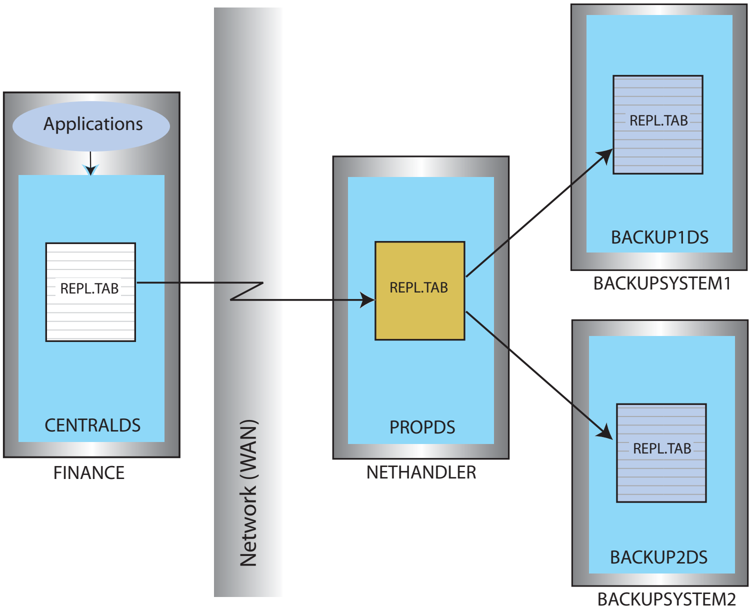
Propagation scheme

Example 3.27 shows a one-way replication scheme from a master data store to a propagator that forwards the changes to two subscribers. For ELEMENT a, the repl.tab table is updated at the centralds data store on the finance machine and replicated to the propds propagator data store on the nethandler machine. For ELEMENT b, the changes to the repl.tab table received by propds are replicated to the two subscribers, backup1ds and backup2ds, on their respective machines, backupsystem1 and backupsystem2.

Example 3.28 provides a similar configuration, but it uses two replication schemes instead of one.

Note that replication for the repl.tab table must be described with separate element names (a and b) in the same scheme, but can be described with the same element name (a) when using separate schemes.

Figure 3.11 Propagation



Example 3.27

```
CREATE REPLICATION repl.propagator
ELEMENT a TABLE repl.tab
  MASTER centralds ON "finance"
  SUBSCRIBER propds ON "nethandler"
ELEMENT b TABLE repl.tab
  PROPAGATOR propds ON "nethandler"
  SUBSCRIBER backup1ds ON "backupsystem1",
             backup2ds ON "backupsystem2";
```

Example 3.28

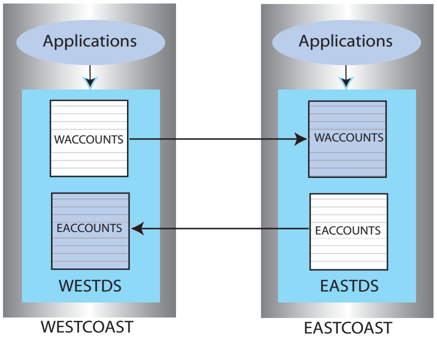
```
CREATE REPLICATION repl.propagator
ELEMENT a TABLE repl.tab
  MASTER centralds ON "finance"
  SUBSCRIBER propds ON "nethandler";

CREATE REPLICATION repl.propagator2
ELEMENT a TABLE repl.tab
  PROPAGATOR propds ON "nethandler"
  SUBSCRIBER backup1ds ON "backupsystem1",
             backup2ds ON "backupsystem2";
```

Bidirectional split workload scheme

Figure 3.12 shows a *split workload* bidirectional replication scheme for two data stores, westds on the westcoast host and eastds on the eastcoast host. Customers are represented in two tables: waccounts contains data for customers in the Western region and eaccounts has data for customers from the Eastern region. The westds data store updates the waccounts table and replicates it to the eastds data store. The eaccounts table is owned by the eastds data store and is replicated to the westds data store. The RETURN RECEIPT attribute enables the return receipt service to guarantee that transactions on either master table are received by their subscriber.

Example 3.30 shows the same configuration using separate replication schemes, r1 and r2.

Figure 3.12 Split workload replication



Example 3.29

```
CREATE REPLICATION repl.r1
ELEMENT elem_waccounts TABLE repl.waccounts
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast" RETURN RECEIPT
ELEMENT elem_eaccounts TABLE repl.eaccounts
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast" RETURN RECEIPT;
```

Example 3.30

```
CREATE REPLICATION repl.r1
ELEMENT elem_waccounts TABLE repl.waccounts
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast" RETURN RECEIPT;

CREATE REPLICATION repl.r2
ELEMENT elem_eaccounts TABLE repl.eaccounts
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast" RETURN RECEIPT;
```

Bidirectional general workload scheme

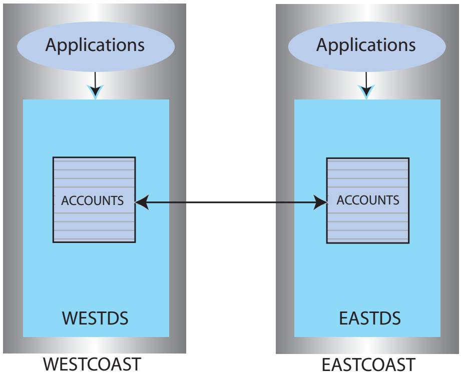
Figure 3.13 shows a *general workload* bidirectional replication scheme in which the `accounts` table can be updated on either the `eastds` or `westds` data store. Each data store is both a master and a subscriber for the `accounts` table.

When elements are replicated in this manner, your applications should write to each data store in a coordinated manner to avoid simultaneous updates on the same data. To manage update conflicts, you can include a timestamp column of type BINARY(8) in your table (as shown by the `tstamp` column in Example 3.32) and enable timestamp comparison by using the replication scheme shown in Example 8.2 on page 172.

See "Replication conflict detection and resolution" on page 167 for a complete discussion on how to manage update conflicts.

Note: A general workload configuration should not be used with the return twosafe return service, nor should it be used with Cache Connect to Oracle.

Figure 3.13 Distributed workload replication



Example 3.31

```
CREATE REPLICATION repl.r1
ELEMENT elem_accounts_1 TABLE repl.accounts
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast"
ELEMENT elem_accounts_2 TABLE repl.accounts
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast";
```

Example 3.32

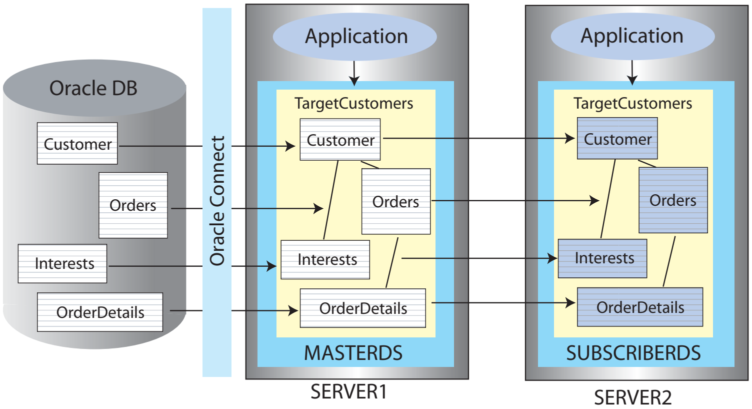
```
CREATE TABLE repl.accounts (custname VARCHAR2(30) NOT NULL,
                             address VARCHAR2(80),
                             curbalance DEC(15,2),
                             tstamp BINARY(8),
                             PRIMARY KEY (custname));
```

Cache group replication scheme

Figure 3.14 shows a database server running Oracle and two application servers, `server1` and `server2`, running TimesTen. The `TargetCustomers` cache group shown in Example 3.34 is replicated in a unidirectional manner from the `masterds` data store running on `server1` to the `TargetCustomers` cache group shown in Example 3.35 in the `subscriberds` data store running on `server2`.

Note: Though the replication scheme definition is shown before the cache group definitions in the example, you must create these cache groups in their respective data stores *before* you apply the replication scheme.

Figure 3.14 Replicating a cache group



Example 3.33 This example shows a scheme that replicates the read-only cache group, `TargetCustomers`, to another read-only cache group.

```
CREATE REPLICATION repl.reptargetcustomers
ELEMENT root TABLE repl.customer
   MASTER masterds ON "server1"
   SUBSCRIBER subscriberds ON "server2"
ELEMENT childorders TABLE repl.orders
   MASTER masterds ON "server1"
   SUBSCRIBER subscriberds ON "server2"
ELEMENT childorderdetails TABLE repl.orderdetails
   MASTER masterds ON "server1"
   SUBSCRIBER SUBSCRIBERDS ON "SERVER2"
ELEMENT childinterests TABLE repl.interests
   MASTER masterds ON "server1"
   SUBSCRIBER subscriberds ON "server2";
```

Example 3.34 This example demonstrates replication of the cache group with AUTOREFRESH STATE ON.

```
CREATE READONLY CACHE GROUP TargetCustomers
AUTOREFRESH INTERVAL 240 MINUTES
FROM
customer(custid NUMBER NOT NULL,
       name CHAR(100) NOT NULL,
       addr CHAR(100),
       zip NUMBER,
       region CHAR(10),
       PRIMARY KEY (custid)),
orders(orderid NUMBER NOT NULL,
       custid NUMBER NOT NULL,
       PRIMARY KEY (orderid),
       FOREIGN KEY (custid) REFERENCES customer(custid)),
orderdetails(orderid NUMBER NOT NULL,
       itemid NUMBER NOT NULL,
       quantity NUMBER NOT NULL,
       PRIMARY KEY (orderid, itemid),
       FOREIGN KEY (orderid) REFERENCES orders(orderid)),
INTERESTS(custid NUMBER NOT NULL,
       interest NUMBER NOT NULL,
       PRIMARY KEY (custid, interest),
       FOREIGN KEY (custid) REFERENCES customer(custid));
```

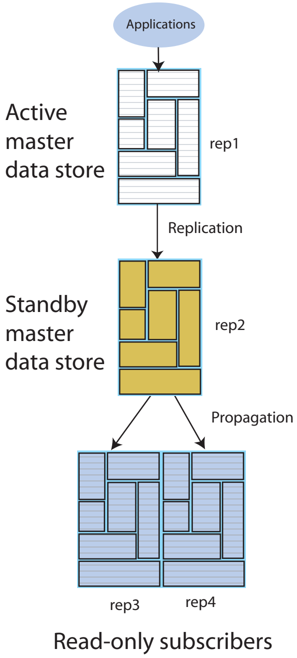
Example 3.35 This example demonstrates replication of the cache group with AUTOREFRESH STATE OFF.

```
CREATE READONLY CACHE GROUP TargetCustomers
AUTOREFRESH STATE OFF
FROM
customer(custid NUMBER NOT NULL,
       name CHAR(100) NOT NULL,
       addr CHAR(100),
       zip NUMBER,
       region CHAR(10),
       PRIMARY KEY (custid)),
orders(orderid NUMBER NOT NULL,
       custid NUMBER NOT NULL,
       PRIMARY KEY (orderid),
       FOREIGN KEY (custid) REFERENCES customer(custid)),
orderdetails(orderid NUMBER NOT NULL,
       itemid NUMBER NOT NULL,
       quantity NUMBER NOT NULL,
       PRIMARY KEY (orderid, itemid),
       FOREIGN KEY (orderid) REFERENCES orders(orderid)),
interests(custid NUMBER NOT NULL,
       interest NUMBER NOT NULL,
       PRIMARY KEY (custid, interest),
       FOREIGN KEY (custid) REFERENCES customer(custid));
```

Active standby pair

An active standby pair is shown in Figure 3.15.

Figure 3.15 Active standby configuration



In an active standby pair, two data stores are defined as masters. One is an active master data store, and the other is a standby master data store. The active master data store is updated directly. The standby master data store receives the updates from the active master data store and propagates the changes to up to 127 read-only subscriber data stores.

In Example 3.36, `rep1` and `rep2` are designated as the master data stores. `rep3` and `rep4` are designated as the subscriber data stores. The replication mode is return receipt.

Note: To create an active standby pair, use the CREATE ACTIVE STANDBY PAIR statement. For more details about setting up an active standby pair, see "Setting up an active standby pair" on page 154.

Example 3.36

```
CREATE ACTIVE STANDBY PAIR rep1 on "node1", rep2 on "node2"
  RETURN RECEIPT
  SUBSCRIBER rep3 ON "node3", rep4 ON "node4";
```

Creating replication schemes with scripts

Creating your replication schemes with scripts can save you time and help you avoid mistakes. This section provides some suggestions for automating the creation of replication schemes using Perl.

Consider the general-workload bidirectional scheme shown in Example 3.37. Entering the ELEMENT description for the five tables, `repl.accounts`, `repl.sales`, `repl.orders`, `repl.inventory`, and `repl.customer`, would be tedious and error-prone if done manually.

Example 3.37

```
CREATE REPLICATION repl.bigscheme
 ELEMENT elem_accounts_1 TABLE repl.accounts
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast"
 ELEMENT elem_accounts_2 TABLE repl.accounts
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast"
 ELEMENT elem_sales_1 TABLE repl.sales
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast"
 ELEMENT elem_sales_2 TABLE repl.sales
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast"
 ELEMENT elem_orders_1 TABLE repl.orders
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast"
 ELEMENT elem_orders_2 TABLE repl.orders
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast"
 ELEMENT elem_inventory_1 TABLE repl.inventory
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast"
 ELEMENT elem_inventory_2 TABLE repl.inventory
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast"
 ELEMENT elem_customers_1 TABLE repl.customers
   MASTER westds ON "westcoast"
   SUBSCRIBER eastds ON "eastcoast"
 ELEMENT elem_customers_2 TABLE repl.customers
   MASTER eastds ON "eastcoast"
   SUBSCRIBER westds ON "westcoast";
```

It is often more convenient to automate the process of writing a replication scheme with scripting. For example, the Perl script shown in Example 3.38 can be used to build the scheme shown in Example 3.37.

Example 3.38

```
@tables = qw(
   repl.accounts
   repl.sales
   repl.orders
   repl.inventory
   repl.customers
 );

print "CREATE REPLICATION repl.bigscheme";

foreach $table (@tables) {
  $element = $table;
  $element =~ s/repl\./elem\_/;

  print "\n";
  print " ELEMENT $element\_1 TABLE $table\n";
  print "   MASTER westds ON \"westcoast\"\n";
  print "   SUBSCRIBER eastds ON \"eastcoast\"\n";
  print " ELEMENT $element\_2 TABLE $table\n";
  print "   MASTER eastds ON \"eastcoast\"\n";
  print "   SUBSCRIBER westds ON \"westcoast\"";
 }
print ";\n";
```

The @tables array shown in Example 3.38 can be obtained from some other source, such as a data store. For example, you can use **ttIsql** and **grep** in a Perl statement to generate a @tables array for all of the tables in the `WestDSN` data store with the owner name `repl`:

```
@tables = `ttIsql -e "tables; quit" WestDSN
           | grep " REPL\."`;
```

Example 3.39 shows a modified version of the script in Example 3.38 that creates a replication scheme for all of the `repl` tables in the `WestDSN` data store. (Note that some substitution may be necessary to remove extra spaces and line feeds from the **grep** output.)

Example 3.39

```
@tables = `ttIsql -e "tables; quit" WestDSN
           | grep " REPL\."`;

print "CREATE REPLICATION repl.bigscheme";

foreach $table (@tables) {
  $table =~ s/^\s*//;   # Remove extra spaces
  $table =~ s/\n//;     # Remove line feeds
  $element = $table;
  $element =~ s/repl\./elem\_/;

  print "\n";
  print " ELEMENT $element\_1 TABLE $table\n";
  print "   MASTER westds ON \"westcoast\"\n";
  print "   SUBSCRIBER eastds ON \"eastcoast\"\n";
  print " ELEMENT $element\_2 TABLE $table\n";
  print "   MASTER eastds ON \"eastcoast\"\n";
  print "   SUBSCRIBER westds ON \"westcoast\"";
 }
print ";\n";
```

Setting Up a Replicated System

This chapter describes how to set up and start replication. The typical tasks related to setting up and starting a replicated system are listed in Table 4.1.

Table 4.1 Tasks related to setting up and starting a replicated system

Task	What you do
Configure the network	See "Configuring the network" on page 100
Establish data stores and set up environment	See "Setting up the replication environment" on page 104
Set security on replicated data stores	See "Replicating access controlled data stores" on page 110
Define a replication scheme	See Chapter 3, "Defining Replication Schemes"
Apply replication scheme to the data stores	See "Applying a replication scheme to a data store" on page 111
Start and stop the replication agent for each data store	See "Starting and stopping the replication agents" on page 111
Set the replication state of subscribers	See "Setting the replication state of subscribers" on page 114

Note: To set up an active standby pair, see "Setting up an active standby pair" on page 154 in Chapter 7, "Administering an Active Standby Pair."

Configuring the network

This section describes some of the issues to be considered when replicating TimesTen data over a network. The general topics are:

- Network bandwidth requirements
- Replication in a WAN environment
- Configuring host IP addresses
- Identifying the local host of a replicated data store

Network bandwidth requirements

The network bandwidth required for TimesTen replication depends on the bulk and frequency of the data being replicated. This discussion explores the types of transactions that characterize the high and low ends of the data range and the network bandwidth required to replicate the data between TimesTen data stores.

The high end of the data range can be characterized by updates or inserts of small amounts of data, such as inserting 128 bytes into a row, which can result in approximately 1.5 - 1.6 MB per second of replicated data. The lower end might be characterized by a single char(10) column update running with return receipt, which can result in approximately 125 KB per second of replicated data.

The following table provides guidelines for calculating the size of replicated records.

Record Type	Size
Begin transaction	48 bytes
Propagate	48 bytes
Update	116 bytes + 18 bytes per column updated + size of old column values + size of new column values + size of the primary key or unique key
Delete	104 bytes + size of the primary key or unique key
Insert	104 bytes + size of the primary key or unique key + size of inserted row
End transaction	48 bytes

Transactions are sent between replicated data stores in batches. A batch is created whenever there is no more data in the transaction log buffer in the master data store, or when the current batch is roughly 256K bytes. At the end of each batch, the master sends a 48-byte end-of-batch message and waits for a 48-byte acknowledgement from the subscriber when the batch is received. See "How replication agents copy updates between data stores" on page 12 for more information.

As shown in the table below, the 100 Base-T Ethernet typical in a LAN can sustain speeds of around 10 MB per second, which is more than enough sustained bandwidth for the most demanding replication rates. However, if servers are communicating in a WAN, the configuration of the replication scheme and transaction load must be carefully matched to the available bandwidth of the network.

Network Area	Network	Sustained Speed
LAN	100 Base-T Ethernet	10 MB per second
WAN	T3	4.8 MB per second
WAN	T2	780 KB per second
WAN	T1	197 KB per second

As shown in the above table, with an available bandwidth of 4.8 MB per second, a T3 line should provide sufficient bandwidth to support 2 subscribers operating at the fastest possible transaction rates (totaling 3.2 MB/s) without loss of performance.

In contrast, a T1 line should provide sufficient bandwidth to accommodate return receipt replication for users inserting less than 1 KB into rows.

Replication in a WAN environment

TimesTen replication uses the TCP/IP protocol, which is not optimized for a WAN environment. You can improve replication performance over a WAN by installing a third-party "TCP stack" product. If replacing the TCP stack is not a feasible solution, you can reduce the amount of network traffic that the TCP/IP protocol has to deal with by setting the COMPRESS TRAFFIC attribute in your CREATE REPLICATION statement. See "Compressing replicated traffic" on page 54 for details.

See *Oracle TimesTen In-Memory Database Installation Guide* for information about changing TCP/IP kernel parameters for better performance.

Configuring host IP addresses

In a replication scheme, you need to identify the name of the host machine on which your data store resides. The operating system translates this host name to one or more IP addresses. This section describes how to configure replication so that it uses the correct host names and IP addresses each host machine.

Identifying data store hosts and network interfaces

When specifying the host for a data store in a replication element, you should always use the name returned by the **hostname** command, as replication uses the this same host name to verify that the current host is involved in the replication scheme. Replication schemes may not be created that do not include the current host.

If a host contains multiple network interfaces (with different IP addresses), you should specify which interfaces are to be used by replication using the ROUTE clause. You must specify a priority for each interface. Replication tries to first connect using the address with the highest priority, and if a connection cannot be established, it tries the remaining addresses in order of priority until a connection is established. If a connection to a host fails while using one IP address, replication attempts to re-connect (or fall back) to another IP address, if more than one address has been specified in the ROUTE clause.

Note: Addresses for the ROUTE clause may be specified as either host names or IP addresses. However, if your host has more than one IP address configured for a given host name, you should only configure the ROUTE clause using the IP addresses, in order to ensure that replication uses only the IP addresses that you intend.

See "Replication with multiple network interfaces" on page 55 for more information.

WINDOWS

Host name resolution on Windows

If a replication configuration is specified using host names rather than IP addresses, replication must be able to translate host names of peers into IP addresses. For this to happen efficiently on Windows, make sure each Windows machine is set up to query either a valid WINS server or a valid DNS server that has correct information about the hosts on the network. In the absence of such servers, static HOST-to-IP entries can be entered in either:

```
%windir%\system32\drivers\etc\hosts
```

or

```
%windir%\system32\drivers\etc\lmhosts
```

Without any of these four options, a Windows machine resorts to broadcasting, which is extremely slow, to detect peer nodes.

You may also encounter extremely slow host name resolution if the Windows machine cannot communicate with the defined WINS servers or DNS servers, or if the host name resolution set up is incorrect on those servers. Use the `ping` command to test whether a host can be efficiently located. The `ping` command responds immediately if host name resolution is set up properly.

Note: You must be consistent in identifying a data store host in a replication scheme. Do not identify a host using its IP address for one data store and then use its host name for the same or another data store.

User-specified addresses for TimesTen daemons and subdaemons

By default, the TimesTen main daemon, all subdaemons and all agents use any available address to listen on a socket for requests. You can modify the `ttendaemon.options` file to specify an address for communication among the agents and daemons by including a `-listenaddr` option. See "Managing TimesTen daemon options" in *Oracle TimesTen In-Memory Database Operations Guide* for details.

Suppose that your machine has two NICs whose addresses are 10.10.10.100 and 10.10.11.200. The loopback address is 127.0.0.1. Then keep in mind the following as it applies to the replication agent:

- If you do not set the `-listenaddr` option in the `ttendaemon.options` file, then any process can talk to the daemons and agents.
- If you set `-listenaddr` to 10.10.10.100, then any process on the local host or the 10.10.10 net can talk to daemons and agents on 10.10.10.100. No processes on the 10.10.11 net can talk to the daemons and agents on 10.10.10.100.
- If you set `-listenaddr` to 127.0.0.1, then only processes on the local host can talk to the daemons and agents. No processes on other hosts can talk the daemons and agents.

Identifying the local host of a replicated data store

Ordinarily, TimesTen replication is able to identify the hosts involved in a replication configuration using normal operating system host name resolution methods. However, in some rare instances, if the host has an unusual host name configuration, TimesTen is unable to determine that the local host matches the host name as specified in the replication scheme. When this occurs, you receive error 8191, “This store is not involved in a replication scheme,” when attempting to start replication using **ttRepStart** or **ttAdmin** `-repStart`. The built-in procedure **ttHostNameSet** may be used in this instance to explicitly indicate to TimesTen that the current data store is in fact the data store specified in the replication scheme. See "ttHostNameSet" in the *Oracle TimesTen In-Memory Database API Reference Guide* for more information.

Setting up the replication environment

The topics related to setting up your replication environment include:

- Establishing the data stores
- Managing the log on a replicated data store
- Managing the log on a replicated data store

Establishing the data stores

You can replicate one or more tables on any existing data store. If the data store you want to replicate does not yet exist, you must first create one, as described in Chapter 1, "Creating TimesTen Data Stores" in *Oracle TimesTen In-Memory Database Operations Guide*.

After you have identified or created the master data store, create a DSN definition for the subscriber data store on the receiving machine. Set the DSN attributes for the master and subscriber data stores as described in "Data store attributes" below.

After you have defined the DSN for your subscriber, you can populate the subscriber data store with the tables to be replicated from the master in one of two ways:

- Connect to the data store and use SQL statements to create new tables in the subscriber data store that match those to be replicated from the master.
- Use the **ttRepAdmin** `-duplicate` utility to copy the entire contents of the master data store to the subscriber, as described in "Copying a master data store to a subscriber" on page 105.

Data store attributes

Replicated data stores must have the following attribute settings in their DSN definitions:

- **Logging**: See "Managing the log on a replicated data store" on page 107 for more information.
- **LogBuffSize** and **LogFileSize**: See "Managing the log on a replicated data store" on page 107.

In addition, data stores which replicate to each other must all have the same **DatabaseCharacterSet** attribute. TimesTen does not perform any character set conversion between replicated data stores.

Note: It is possible to replicate between data stores with different settings for the **TypeMode** attribute. However, you must make sure that the underlying data type for each replicated column is the same on each node. See "TypeMode" on page 21 of the *Oracle TimesTen In-Memory Database API Reference Guide* for more information.

Table requirements and restrictions

Tables to be replicated in any type of replication scheme must have the following characteristics:

- The name, owner, and column definitions of the tables participating in the replication scheme must be identical on both the master and subscriber data stores.
- Tables to be replicated must have one of the following:
  - A primary key
  - A unique index over non-nullable columns

  Replication uses the primary key or unique index to uniquely identify each row in the replicated table. Replication always selects the first usable index that turns up in a sequential check of the table's index array. If there is no primary key, replication selects the first unique index without NULL columns it encounters. The selected index on the replicated table in the master data store must also exist on its counterpart table in the subscriber.

Note: The keys on replicated tables are transported in each update record to the subscribers. Smaller keys transport most efficiently.

- VARCHAR2, NVARCHAR2, VARBINARY and TT_VARCHAR columns in replicated tables must be limited to a size of 256,000 bytes. For a VARCHAR2 column, the maximum length when using character length semantics depends on the number of bytes each character occupies when using a particular data store character set. For example, if the character set requires four bytes for each character, the maximum possible length is 64,000 characters. For an NVARCHAR2 column, which requires two bytes for each character, the maximum length when using character length semantics is 128,000 characters.
- If Access Control is enabled, you must have privileges to CREATE REPLICATION, DROP TABLE, INSERT, SELECT, UPDATE, DELETE for the tables that are replicated.
- Temporary tables can be defined and used in a data store that has a replication scheme defined, but temporary tables themselves cannot be replicated.

Copying a master data store to a subscriber

A shorthand method for populating a subscriber data store that is to fully replicate its master data store is to simply copy the contents of the master. Copying a data store in this manner is also essential when recovering a failed data store, as described in "Managing data store failover and recovery" on page 183.

You can use either the **ttRepAdmin** `-duplicate` utility or the **ttRepDuplicateEx** C function to duplicate a data store. However, before copying the contents of a master data store to populate a subscriber data store, you must:

1. Create a DSN for the new subscriber data store.
2. Create or alter a replication scheme to include the new subscriber data store and its host, as described in "Defining a replication scheme" on page 44.
3. Apply the replication scheme to the master data store, as described in "Applying a replication scheme to a data store" on page 111.
4. Start the replication agent for the master data store, as described in "Starting and stopping the replication agents" on page 111.

For example, on host `server1`, we have a DSN named `masterDSN` that describes the `masterds` data store. On host `server2`, we have a DSN named `newstoreDSN` that describes the `newstore` data store.

To populate the `newstore` data store with the contents of `masterds`, perform the following tasks:

On server1:

Using a text editor, create a new SQL file, named `newrepscheme.sql`, that defines the replication scheme and calls the **ttRepStart** procedure to start replication:

```
CREATE REPLICATION repl.repscheme
  ELEMENT e TABLE repl.tab
  MASTER masterds ON "server1"
  SUBSCRIBER newstore ON "server2";
call ttRepStart;
```

From the command line, configure `masterds` with the replication scheme and start the replication agent:

```
> ttIsql -f newrepscheme.sql masterds
```

On server2:

From the command line, copy the contents of the `masterds` data store into the `newstore` data store:

```
> ttRepAdmin -dsn newstore -duplicate -from masterds
-host "server1"
```

The `newstore` data store should now have the same contents as the `masterds` data store.

Note: The `-host` can be identified with either the name of the remote host or its TCP/IP address. If you identify hosts using TCP/IP addresses, you must identify the address of the local host (`server2` in this example) by using the `-localhost` option. For details, see "**ttRepAdmin**" in the *Oracle TimesTen In-Memory Database API Reference Guide*.

You can also do a duplication operation similar to that shown above from a C program by using the **ttRepStart** procedure and **ttRepDuplicateEx** C function. See "Starting and stopping the replication agents" on page 111 and "Recovering a failed data store" on page 187 for more information.

Problems? For the latest troubleshooting information, see "Troubleshooting Replication" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

Managing the log on a replicated data store

This section includes the following topics:

- About log buffer size and persistence
- About log growth on a master data store
- Setting the log failure threshold
- Setting attributes for disk-based logging

About log buffer size and persistence

A common misconception among TimesTen users is that there is a relationship between the size of the log buffer and lost transactions. The size of the log buffer has no impact on persistence.

If your DSN is configured with **DurableCommits**=0, then transactions are written durably to disk only under the following circumstances:

- When the log buffer fills up.
- When a **ttDurableCommit** is called or when a transaction on a connection with **DurableCommits**=1 is committed or rolled back.
- When the replication agent sends a batch of transactions to a subscriber and the master has been configured for replication with the TRANSMIT DURABLE attribute (the default). (See "Default replication" on page 12.)
- When the replication agent periodically executes a durable commit, whether the primary store is configured with TRANSMIT DURABLE or not.
- When your DSN is configured with **LogFlushMethod**=2, writes are written to disk before control is returned to the application.

The size of the log buffer has no influence on the ability of TimesTen to write data to disk under any of the circumstances listed above.

About log growth on a master data store

With data stores that do not use replication, XLA, Cache Connect, or incremental backup, unneeded records in the log buffer and unneeded log files are purged each time the application calls a **ttCkpt** or **ttCkptBlocking** procedure. With a replicated data store, transactions remain in the log buffer and log files until the master replication agent confirms they have been fully processed by the subscriber, as described in "How replication works" on page 11. Only then can the master consider purging them from the log buffer and log files.

A master data store log can grow much larger than it would on an unreplicated data store if there are changes to its subscriber state (see "Setting the replication state of subscribers" on page 114 for information on the subscriber states). When the subscriber is in the **Start** state, the master can purge logged data after it receives confirmation it has been received by the subscriber. However, if a subscriber becomes unavailable or set to the **Pause** state, the log on the master data store cannot be flushed and the space used for logging can be exhausted. When the log space is exhausted, subsequent updates on the master data store are aborted.

Setting the log failure threshold

You can establish a *threshold* value that, when exceeded, sets an unavailable subscriber to the **Failed** state before the available log space is exhausted. You can set the log threshold by specifying a STORE parameter with a FAILTHRESHOLD value in your CREATE REPLICATION or ALTER REPLICATION statement. (See Example 3.23 on page 85 for an example.)

Note: If you use ALTER REPLICATION to reset the threshold value on an existing replication scheme, you must first stop the replication agents before using the ALTER REPLICATION to define a new threshold value, and then restart the replication agents.

The default threshold value is 0, which means "no limit." See "Setting attributes for disk-based logging" on page 109 for details.

If a master sets a subscriber data store to the **Failed** state, it drops all of the data for the failed subscriber from its log and transmits a message to the failed subscriber data store. (If the master replication agent can communicate with the subscriber replication agent, then the message is transmitted immediately. Otherwise, the message is transmitted when the connection is reestablished.) After receiving the message from the master, if the subscriber is configured for bidirectional replication or to propagate updates to other subscribers, it does not transmit any further updates, because its state from a replication standpoint has been compromised.

Any application that connects to the failed subscriber receives a *tt_ErrReplicationInvalid* (8025) warning indicating that the data store has been marked **Failed** by a replication peer. Once the subscriber data store has been informed of its failed status, its state on the master data store is changed from **Failed** to **Stop**.

Applications can use the ODBC **SQLGetInfo** function to check if the data store it is connected to has been set to the **Failed** state, as described in "Subscriber failures" on page 184.

Setting attributes for disk-based logging

The **LogBuffSize** specifies the maximum size of your in-memory log buffer. This buffer is flushed to a log file on the disk when it becomes full. Smaller **LogBuffSize** values may impact performance, but not reliability.

When logging to disk, your main concern is establishing enough disk space for the replication log files. There are two settings that control the amount of disk space used by your log:

- The **LogFileSize** setting in your DSN specifies the maximum size of a log file. Should your logging requirements exceed this value, additional log files with the same maximum size are created. (If you set the **LogFileSize** to a smaller value than the **LogBuffSize**, TimesTen automatically increases the **LogFileSize** to match the **LogBuffSize**.)
- The log *threshold* setting specifies the maximum number of *log files* allowed to accumulate before the master assumes a subscriber has failed. The threshold value is the number of log files *between* the most recently written to log file and the earliest log file being held for the subscriber. For example, if the last record successfully received by all subscribers was in Log File 1 and the last log record written to disk is at the beginning of Log File 4, then replication is at least 2 log files behind (the contents of Log Files 2 and 3). If the threshold value is 2, then the master sets the subscriber to the **Failed** state after detecting the threshold value had been exceeded. This may take up to 10 seconds. See "Setting the log failure threshold" on page 108 for more information.

When transactions are logged to disk, you can use *bookmarks* to detect the LSNs (*log sequence numbers*) of the update records that have been replicated to subscribers and those that have been written to disk. To view the location of the bookmarks for the subscribers associated with *masterDSN*, use the c utility or **ttBookmark** procedure, as described in "Show replicated log records" on page 130.

If a subscriber goes down and then comes back up before the threshold is reached, then replication automatically "catches up" as the committed transactions in the log files following the bookmark are automatically transmitted. However, if the threshold is exceeded, the master sets the subscriber to the **Failed** state. A failed subscriber must use **ttRepAdmin** `-duplicate` to copy the master data store and start over, as described in "Managing data store failover and recovery" on page 183.

Configuring a large number of subscribers

A replication scheme can include up to 128 subscribers. An active standby pair can include up to 127 read-only subscribers. If you are planning a replication scheme that includes a large number of subscribers, then ensure the following:

- The log buffer size should result in the value of LOG_FS_READS in the SYS.MONITOR table being 0 or close to 0. This ensures that the replication agent does not have to read any log records from disk. If the value of LOG_FS_READS is increasing, then increase the log buffer size.
- CPU resources are adequate. The replication agent on the master data store spawns a thread for every subscriber data store. Each thread reads and processes the log independently and needs adequate CPU resources to make progress.

Replicating access controlled data stores

When a data store is installed with Access Control enabled, replication daemon administration is restricted to users with the ADMIN privilege and the ability to create and change replication schemas is restricted to users with the DDL privilege. However, replicated updates from the master are applied to a subscriber regardless of access controls present on the subscriber. This means you can enable Access Control on a master and not on a subscriber, or the other way around.

In general, you should configure similar Access Controls on all instances in a replication schema to avoid confusion. In some configurations it might make sense to control access to the master and not the subscriber(s). That way you can ensure the integrity of the data on the master but still provide global access to the data on the subscribers. For example, you can replicate data from a secure master to both a secure subscriber (for recovery) and a non-secure subscriber that can be read by everyone.

See Chapter 1, “Access Control” in the *Oracle TimesTen In-Memory Database Installation Guide* for more information on how to install and configure TimesTen Access Control.

Replicating data stores across releases

Replication functions across releases only if the data store of the more recent version of TimesTen was upgraded using **ttMigrate** from a data store of the older version of TimesTen. A data store created in the more recent version of TimesTen is not guaranteed to replicate correctly with the older version.

For example, replication between a data store created in a 5.1 version of TimesTen and a data store created in a 6.0 version of TimesTen is not supported. However, if one data store was created in a 5.1 version, and the peer data store was created in a 5.1 version and then upgraded to a 6.0 version, replication between them is supported.

Applying a replication scheme to a data store

Define your replication scheme as described in Chapter 3, “Defining Replication Schemes.” Save the CREATE REPLICATION statement in a SQL file.

After you have described your replication scheme in a SQL file, you can execute the SQL on the data store using the `-f` option to the **ttIsql** utility. The syntax is:

```
ttIsql -f schemefile.sql -connstr "dsn=DSN"
```

Example 4.1 If your replication scheme is described in a file called `repscheme.sql`, you can execute the file on a DSN, called *masterDSN*, by entering:

```
> ttIsql -f repscheme.sql -connstr "dsn=masterDSN"
```

Under most circumstances, you should apply the same scheme to all of your replicated data stores. You must invoke a separate **ttIsql** command on each host to apply the replication scheme.

Example 4.2 If your scheme includes the data stores `masterDSN` on host `S1`, `subscriber1DSN` on host `S2`, and `subscriber2DSN` on host `S3`, do the following:

On host S1, enter:

```
> ttIsql -f repscheme.sql masterDSN
```

On host S2, enter:

```
> ttIsql -f repscheme.sql subscriber1DSN
```

On host S3, enter:

```
> ttIsql -f repscheme.sql subscriber2DSN
```

You can also execute the SQL file containing your replication scheme from the **ttIsql** command line. For example:

```
Command> run repscheme.sql;
```

Starting and stopping the replication agents

After you have defined a replication scheme, you can start the replication agents for each data store involved in the replication scheme.

Note: If TimesTen was installed with Access Control enabled, you must have ADMIN privileges to the data store to start or stop a replication agent. See Chapter 1, “Access Control” in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

You can start and stop replication agents from either the command line or from your program, as described in the sections:

- Controlling replication agents from the command line
- Controlling replication agents from a program

Note: If a data store does not participate in a replication scheme, attempts to start a replication agent for that data store fail.

Controlling replication agents from the command line

To start and stop a replication agent from the command line, use the **ttAdmin** utility with the `-repStart` or `-repStop` option:

```
ttAdmin -repStart DSN
ttAdmin -repStop DSN
```

Note: Replication DDL that is not permitted when the replication agent is running may be possible during the brief period of time between issuing **ttAdmin** `-repStart` command and the actual start of the replication agent. For example, it may be possible to drop a replication scheme during this time.

Example 4.3

To start the replication agents for the DSNs named `masterDSN` and `subscriberDSN`, enter:

```
ttAdmin -repStart masterDSN
ttAdmin -repStart subscriberDSN
```

To stop the replication agents, enter:

```
ttAdmin -repStop masterDSN
ttAdmin -repStop subscriberDSN
```

You can also use the **ttRepStart** and **ttRepStop** procedures to start and stop a replication agent from the **ttIsql** command line.

Example 4.4

To start and stop the replication agent for the DSN named `masterDSN`, enter:

```
> ttIsql masterDSN
Command> call ttRepStart;
Command> call ttRepStop;
```

You can also use the **ttAdmin** utility to set the *replication restart policy*. By default the policy is `manual`, which enables you to start and stop the replication agents as described above. Alternatively, you can set the replication restart policy for a data store to `always` or `norestart`.

Table 4.2 Replication Restart Policies

Restart Policy	Start replication agent when the TimesTen daemon starts	Restart replication agent on errors or invalidations
`always`	Yes	Yes
`manual`	No	Yes
`norestart`	No	No

Note: The replication agents are managed by the TimesTen daemon, which must be started before starting any replication agents.

When the restart policy is `always`, the replication agent is automatically started when the data store is loaded into memory. (See "Specifying a RAM policy" in the *Oracle TimesTen In-Memory Database Operations Guide* to determine when a data store is loaded into memory.)

Example 4.5 To use **ttAdmin** to set the replication restart policy to `always`, enter:

```
ttAdmin -repPolicy always DSN
```

To reset the policy back to manual, enter:

```
ttAdmin -repPolicy manual DSN
```

Following an error or data store invalidation, both `manual` and `always` policies cause the replication agent to be automatically restarted. When the agent restarts automatically, it is often the first connection to the data store. This happens after a fatal error that, for example, requires all applications to disconnect. The first connection to a data store usually has to load the most recent checkpoint file and often needs to do recovery. For a very large data store, this process may take several minutes. During this period, all activity on the data store is blocked so that new connections cannot take place and any old connections cannot finish disconnecting. This may also result in two copies of the data store existing at the same time because the old one stays around until all applications have disconnected. For very large data stores for which the first-connect time may be significant, you may want to wait for the old data store to become inactive first before starting up the new one. You can do this by setting the restart policy to `norestart` to specify that the replication agent is not to be automatically restarted.

Controlling replication agents from a program

To start and stop the replication agent for a data store from your program, connect to the replicated data store and use the **ttRepStart** and **ttRepStop** procedures.

Example 4.6

To start and stop the replication agent for the data store that is identified by the hdbc connection handle:

```
rc = SQLAllocStmt( hdbc, &hstmt );
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepStart()", SQL_NTS );
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepStop()", SQL_NTS );
```

You can programmatically set the replication restart policy by calling the **ttRepPolicySet** procedure. You can use this procedure to set the restart policy for a data store to either `manual` (default), `always`, or `norestart` in the same manner as described for **ttAdmin** `-repPolicy` in "Controlling replication agents from the command line" on page 112.

Example 4.7

To set the replication policy to `always` for the data store identified by the hdbc connection handle:

```
rc = SQLAllocStmt( hdbc, &hstmt );
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepPolicy ('always')", SQL_NTS );
```

Setting the replication state of subscribers

The "state" of a subscriber replication agent is described by its master data store. When recovering a failed subscriber data store, you must reset the replication state of the subscriber data store with respect to the master data store it communicates with in a replication scheme. You can reset the state of a subscriber data store from either the command line or your program:

- From the command line, use **ttRepAdmin** `-state` to direct a master data store to reset the replication state of one of its subscriber data stores.
- From your program, invoke the **ttRepSubscriberStateSet** procedure to direct a master data store to reset the replication state of one or all of its subscriber data stores.

See Chapter 5, "Monitoring Replication" for information on how to query the state of a data store.

A master data store can set a subscriber data store to either the **Start**, **Pause**, or **Stop** states. The data store state appears as an integer value in the STATE column in the TTREP.REPNETWORK table, as shown below.

State	Description
Start STATE Value: 0	Replication updates are collected and transmitted to the subscriber data store as soon as possible. If replication for the subscriber data store is not operational, the updates are saved in the log files until they can be sent.
Pause STATE Value: 1	Replication updates are retained in the log with no attempt to transmit them. Transmission begins when the state is changed to **Start**.
Stop STATE Value: 2	Replication updates are discarded without being sent to the subscriber data store. Placing a subscriber data store in the **Stop** state discards any pending updates from the master's transaction log.
Failed STATE Value: 4	Replication to a subscriber is considered failed because the threshold limit (log data) has been exceeded. This state is set by the system is a transitional state before the system sets the state to **Stop**. Applications that connect to a Failed data store receive a warning. See "General failover and recovery procedures" on page 184 for more information.

When a master data store sets one of its subscribers to the **Start** state, updates for the subscriber are retained in the master's log. When a subscriber is in the **Stop** state, updates intended for it are discarded.

When a subscriber is in the **Pause** state, updates for it are retained in the master's log, but are not transmitted to the subscriber data store. When a master transitions a subscriber from **Pause** to **Start**, the backlog of updates stored in the master's log is transmitted to the subscriber. (There is an exception to this, which is described in "Managing data store failover and recovery" on page 183.) If a master data store is unable to establish a connection to a stated subscriber, the master periodically attempts to establish a connection until successful.

Example 4.8 To use **ttRepAdmin** from the command line to direct the `masterds` master data store to set the state of the `subscriberds` subscriber data store to **Stop**:

```
ttRepAdmin -dsn masterds -receiver -name subscriberds -state stop
```

Note: If you have multiple subscribers with the same name on different hosts, use the **ttRepAdmin** `-host` parameter to identify the host for the subscriber.

Example 4.9 Assuming the replication scheme is named `repl.scheme`, the following **ttRepSubscriberStateSet** procedure directs the master data store to set the state of the subscriber data store (`subscriberds ON system1`) to **Stop**:

```
rc = SQLAllocStmt( hdbc, &hstmt );
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepSubscriberStateSet('repscheme', 'repl',
     'subscriberds', 'system1', 2)", SQL_NTS );
```

Example 4.10 The following **ttRepSubscriberStateSet** procedure directs the master data store to set the state of all of its subscriber data stores to **Pause**:

```
rc = SQLAllocStmt( hdbc, &hstmt );
rc = SQLExecDirect( hstmt, (SQLCHAR *)
    "CALL ttRepSubscriberStateSet( , , , , 1 )", SQL_NTS );
```

Only **ttRepSubscriberStateSet** can be used to set all of the subscribers of a master to a particular state.The **ttRepAdmin** utility does not have any equivalent functionality.

5

Monitoring Replication

This chapter describes some of the TimesTen utilities and procedures you can use to monitor the replication status of your data stores.

You can monitor replication from both the command line and within your programs. The **ttStatus** and **ttRepAdmin** utilities described in this chapter are useful for command line queries. To monitor replication from your programs, you can use the TimesTen procedures described in Chapter 2, "Built-In Procedures" of the *Oracle TimesTen In-Memory Database API Reference Guide* or create your own SQL SELECT statements to query the replication tables described in Chapter 7, "System and Replication Tables" of the *Oracle TimesTen In-Memory Database SQL Reference Guide*.

Note: The TimesTen SYS and TTREP tables can only be accessed for queries. ***You cannot* directly alter the contents of these tables.**

This chapter includes the following topics:

- Show state of replication agents
- Show master data store information
- Show subscriber data store information
- Show configuration of replicated data stores
- Show replicated log records
- Show replication status
- Show the return service status for a subscriber

Show state of replication agents

You can display information about the current state of the replication agents:

- From the command line: ttStatus
- From the command line: ttAdmin -query
- From a program: ttDataStoreStatus

You can also obtain the state of specific replicated data stores as described in "Show subscriber data store information" on page 122 and "Show configuration of replicated data stores" on page 126.

From the command line: ttStatus

Use the **ttStatus** utility to confirm that the replication agents are started for the master and subscriber data stores. The output from a simple replication scheme using a single master and subscriber data store (such as the scheme described in "Single subscriber scheme" on page 83) should look similar to the output in Example 5.1.

Example 5.1

```
> ttStatus
TimesTen status report as of Mon Dec 13 16:07:09 2004

Daemon pid 568 port 15100 instance tt51
TimesTen server pid 1372 started on port 15102
TimesTen webserver pid 1168 started on port 15104

------------------------------------------------------------------------
Data store c:\temp\subscriberds
There are 7 connections to the data store
Data store is in shared mode
Shared Memory KEY Global\DBI41be2db3.1.SHM.4 HANDLE 0x294
Process      pid 2764 context 0xb9ab70 connected (KEY
Global\DBI41be2db3.1.SHM.4)
Replication  pid 1784 context 0x849008 connected (KEY
Global\DBI41be2db3.1.SHM.4)
Replication  pid 1784 context 0x900008 connected (KEY
Global\DBI41be2db3.1.SHM.4)
Replication  pid 1784 context 0x904f68 connected (KEY
Global\DBI41be2db3.1.SHM.4)
Subdaemon    pid 156 context 0xda0068 connected (KEY Global\DBI41be2db3.1.SHM.4)
Subdaemon    pid 156 context 0xe4bd30 connected (KEY Global\DBI41be2db3.1.SHM.4)
Subdaemon    pid 156 context 0xe5c008 connected (KEY Global\DBI41be2db3.1.SHM.4)
Replication policy  : Manual
Replication agent is running.
Oracle agent policy : Manual
------------------------------------------------------------------------
Data store c:\temp\masterds
There are 8 connections to the data store
Data store is in shared mode
Shared Memory KEY Global\DBI41b8bacb.0.SHM.6 HANDLE 0x2dc
Process      pid 2208 context 0xb9ab70 connected (KEY
Global\DBI41b8bacb.0.SHM.6)
Replication  pid 2708 context 0x849008 connected (KEY
Global\DBI41b8bacb.0.SHM.6)
Replication  pid 2708 context 0x8ebf28 connected (KEY
Global\DBI41b8bacb.0.SHM.6)
Replication  pid 2708 context 0x8fbff8 connected (KEY
Global\DBI41b8bacb.0.SHM.6)
```

```
Replication  pid 2708 context 0x900f58 connected (KEY
Global\DBI41b8bacb.0.SHM.6)
Subdaemon  pid 1120 context 0xda0068 connected (KEY Global\DBI41b8bacb.0.SHM.6)
Subdaemon  pid 1120 context 0xe3bb28 connected (KEY Global\DBI41b8bacb.0.SHM.6)
Subdaemon  pid 1120 context 0xe60008 connected (KEY Global\DBI41b8bacb.0.SHM.6)
Replication policy  : Manual
Replication agent is running.
Oracle agent policy : Manual
```

From the command line: ttAdmin -query

Use the **ttAdmin** utility with the `-query` option to confirm the policy settings for a data store, including the replication restart policy described in "Starting and stopping the replication agents" on page 111.

Example 5.2

```
> ttAdmin -query masterDSN
RAM Residence Policy            : inUse
Manually Loaded In Ram          : False
Replication Agent Policy        : manual
Replication Manually Started    : True
Oracle Agent Policy             : manual
Oracle Agent Manually Started   : False
```

From a program: ttDataStoreStatus

To obtain the status of the replication agents from a program, use the **ttDataStoreStatus** procedure.

Example 5.3

To call **ttDataStoreStatus** within SQL to obtain the status of the replication agents for the `masterds` and `subscriberds` data stores, you could use:

```
> ttIsql masterds
Command> CALL ttDataStoreStatus('/tmp/masterds');
< /tmp/masterds, 964, 00000000005D8150, subdaemon,
Global\DBI3b3234c0.0.SHM.35 >
< /tmp/masterds, 1712, 00000000016A72E0, replication,
Global\DBI3b3234c0.0.SHM.35 >
< /tmp/masterds, 1712, 0000000001683DE8, replication,
Global\DBI3b3234c0.0.SHM.35 >
< /tmp/masterds, 1620, 0000000000608128, application,
Global\DBI3b3234c0.0.SHM.35 >
4 rows found.

Command> CALL ttDataStoreStatus('/tmp/subscriberds');
< /tmp/subscriberds, 956, 00000000005D8150, subdaemon,
Global\DBI3b5c82a2.1.SHM.42 >
< /tmp/subscriberds, 1760, 00000000016B72E8, replication,
Global\DBI3b5c82a2.1.SHM.42 >
```

```
< /tmp/subscriberds, 1760, 0000000001683DE8, replication,
Global\DBI3b5c82a2.1.SHM.42 >
3 rows found.
```

The output from **ttDataStoreStatus** is similar to that shown for the **ttStatus** utility in "From the command line: ttStatus" on page 118.

Example 5.4 You can also call **ttDataStoreStatus** within a **SQLExecDirect** function to obtain the status of the `masterds` replication agent:

```
#define STATUS_LEN 30
UCHAR status[STATUS_LEN];

rc = SQLExecDirect( hstmt, (SQLCHAR *)
"CALL ttDataStoreStatus ('/tmp/masterds')", SQL_NTS );
  if (rc == SQL_SUCCESS) {
   SQLBindCol(hstmt, 4, SQL_C_CHAR, status, STATUS_LEN, &cbStat);
  }
```

Show master data store information

You can display information for a master data store:

- From the command line: ttRepAdmin -self -list
- From a program: SQL SELECT statement

From the command line: ttRepAdmin -self -list

To display information for a master data store from the command line, use the **ttRepAdmin** utility with the `-self -list` options:

```
ttRepAdmin -dsn masterDSN -self -list
```

Example 5.5 This example shows the output for the master data store described in "Multiple subscriber schemes" on page 84.

```
> ttRepAdmin -dsn masterds -self -list
Self host "server1", port auto, name "masterds", LSN 0/2114272
```

See Example 5.6 for a description of each field.

From a program: SQL SELECT statement

To obtain the information for a master data store from a program, use the following SQL SELECT statement to query the TTREP.REPSTORES and TTREP.REPSTORES tables:

```
SELECT t.host_name, t.rep_port_number, t.tt_store_name
 FROM ttrep.ttstores t, ttrep.repstores s
  WHERE t.is_local_store = 0x01
   AND t.tt_store_id = s.tt_store_id;
```

Use the **ttBookmark** procedure to obtain the *replication hold LSN*, as described in "Show replicated log records" on page 130.

Example 5.6 This is the output of the above SELECT statement for the master data store described in "Multiple subscriber schemes" on page 84. The fields are the host name, the replication port number, and the data store name.

```
< server1, 0, masterds>
```

Call the **ttBookmark** procedure to obtain the replication hold LSN.

```
> ttIsql masterds
Command> call ttBookMark();
< 10, 928908, 10, 280540, 10, 927692 >
1 row found.
```

The output fields are defined as follows:

Field	Description
host	The name of the host machine for the data store.
port	TCP/IP port used by a replication agent of another data store to receive updates from this data store. A value of 0 (zero) indicates replication has automatically assigned the port.
name	Name of the data store
Log File / Last written LSN	The log file and log sequence number (LSN) that identify the location of the most recently generated transaction log record for the data store.

Field	Description
Log File / Last LSN forced to disk	The log file and LSN that identify the location of the most recent transaction log record written to the disk.
Log File / Replication hold LSN	The log file and LSN that identify the location of the lowest (or oldest) record held in the log for possible transmission to a subscriber. A value of -1/-1 indicates replication is in the **Stop** state with respect to all subscribers.

Show subscriber data store information

Replication uses the TimesTen transaction log to retain information that must be transmitted to subscriber sites. When communication to subscriber data stores is interrupted or the subscriber sites are down, the log data accumulates. Part of the output from the queries described in this section allows you to see how much log data has accumulated on behalf of each subscriber data store and the amount of time since the last successful communication with each subscriber data store.

You can display information for subscriber data stores:

- From the command line: ttRepAdmin -receiver -list
- From a program: ttReplicationStatus procedure
- From a program: SQL SELECT statement

From the command line: ttRepAdmin -receiver -list

To display information about a master data store's subscribers from the command line, use the **ttRepAdmin** utility with the `-receiver -list` options:

```
ttRepAdmin -dsn masterDSN -receiver -list
```

Example 5.7

This example shows the output for the subscribers described in "Multiple subscriber schemes" on page 84.

```
> ttRepAdmin -dsn masterds -receiver -list
Peer name         Host name                 Port    State  Proto
----------------  ------------------------ ------  ------- -----
subscriber1ds     server2                   Auto    Start     10

Last Msg Sent Last Msg Recv Latency TPS     RecordsPS Logs
------------- ------------- ------- ------- --------- ----
0:01:12       -               19.41       5        52    2

Peer name         Host name                 Port    State  Proto
----------------  ------------------------ ------  ------- -----
subscriber2ds     server3                   Auto    Start     10
```

```
Last Msg Sent Last Msg Recv Latency TPS     RecordsPS Logs
------------- ------------- ------- ------- --------- ----
0:01:04       -               20.94       4        48    2
```

The first line of the display contains the subscriber definition. The following row of the display contains latency and rate information, as well as the number of log files being retained on behalf of this subscriber. See Example 5.10 on page 125 for a description of each field.

Problems? If you have more than one scheme specified in your TTREP.REPLICATIONS table, you must use the -scheme option to specify which scheme you wish to list. Otherwise you receive the following error:

```
Must specify -scheme to identify which replication scheme to use
```

For the latest troubleshooting information, "Troubleshooting Replication" in *Oracle TimesTen In-Memory Database Troubleshooting Procedures Guide*.

From a program: ttReplicationStatus procedure

You can obtain more detailed status for a specific replicated data store from a program by using the **ttReplicationStatus** procedure.

Example 5.8 You can use **ttReplicationStatus** to obtain the replication status of the subscriberds data store in relation to its master data store. From the master data store, enter:

```
> ttIsql masterds
```

```
Command> CALL ttReplicationStatus ('subscriberds');
< subscriberds, myhost, 0, start, 1, 152959, repscheme, repl>
1 row found.
```

See Example 5.9 for an explanation of the output fields.

Example 5.9 You can also call **ttReplicationStatus** within a **SQLExecDirect** function to obtain the replication status of the subscriberds data store:

```
#define STATUS_LEN 30
UCHAR status[STATUS_LEN];

rc = SQLExecDirect( hstmt, (SQLCHAR *)
"CALL ttReplicationStatus ('subscriberds')", SQL_NTS );
  if (rc == SQL_SUCCESS) {
   SQLBindCol(hstmt, 4, SQL_C_CHAR, status, STATUS_LEN, &cbStat);
  }
```

The columns in the returned row are shown in the following table:

Column	Description
Subscriber name	Name of the subscriber data store.
Host name	Name of the machine that hosts the subscriber.
Port	TCP/IP port used by the subscriber agent to receive updates from the master. A value of 0 indicates replication has automatically assigned the port.
State	Current state of the subscriber with respect to its master data store (see "Setting the replication state of subscribers" on page 114 for information).
Logs	Number of log files the master data store is retaining for this subscriber.
Last Msg Sent	Time (in seconds) since the master sent the last message to the subscriber. Note that this includes the "heart beat" messages sent between the data stores.
Replication scheme name	The name of the replication scheme used.
Owner name	The name of the owner of the replication scheme.

From a program: SQL SELECT statement

To obtain information about a master's subscribers from a program, use the following SQL SELECT statement to query the TTREP.REPNETWORK, TTREP.TTSTORES, and SYS.MONITOR tables:

```
SELECT t1.tt_store_name, t1.host_name, t1.rep_port_number,
p.state, p.protocol, p.timesend, p.timerecv, p.latency,
p.tps, p.recspersec, t3.last_log_file - p.sendlsnhigh + 1
  FROM ttrep.reppeers p, ttrep.ttstores t1, ttrep.ttstores t2,
   sys.monitor t3
  WHERE p.tt_store_id = t2.tt_store_id
   AND t2.is_local_store = 0X01
   AND p.subscriber_id = t1.tt_store_id
   AND p.replication_name = 'repscheme'
   AND p.replication_owner = 'repl'
   AND (p.state = 0 OR p.state = 1);
```

Example 5.10 The following is sample output from the SELECT statement above:

```
< subscriber1ds, server2, 0, 0, 7, 1003941635, 0, -1.00000000000000, -1, -1, 1 >
< subscriber2ds, server3, 0, 0, 7, 1003941635, 0, -1.00000000000000, -1, -1, 1 >
```

The output from either the **ttRepAdmin** utility or the SQL SELECT statement contains the following fields:

Field	Description
Peer name	Name of the subscriber data store
Host name	Name of the machine that hosts the subscriber
Port	TCP/IP port used by the subscriber agent to receive updates from the master. A value of 0 indicates replication has automatically assigned the port.
State	Current replication state of the subscriber with respect to its master data store (see "Setting the replication state of subscribers" on page 114 for information).
Proto	Internal protocol used by replication to communicate between this master and its subscribers. You can ignore this value.
Last Msg Sent	Time (in seconds) since the master sent the last message to the subscriber. Note that this includes the "heart beat" messages sent between the data stores.
Last Msg Recv	Time (in seconds) since this subscriber received the last message from the master.

Field	Description
Latency	The average latency time (in seconds) between when the master sends a message and when it receives the final acknowledgement from the subscriber. (See note below.)
TPS	The average number of transactions per second that are committed on the master and processed by the subscriber. (See note below.)
RecordsPS	The average number of transmitted records per second. (See note below.)
Logs	Number of log files the master data store is retaining for a subscriber.

Note: Latency, TPS, and RecordsPS report averages detected while replicating a batch of records. These values can be unstable if the workload is not relatively constant. A value of -1 indicates the master’s replication agent has not yet established communication with its subscriber replication agents or sent data to them.

Show configuration of replicated data stores

You can display the configuration of your replicated data stores:

- From ttIsql: repschemes command
- From the command line: ttRepAdmin -showconfig
- From a program: SQL SELECT statements

From ttIsql: repschemes command

To display the configuration of your replicated data stores from the **ttIsql** prompt, use the `repschemes` command:

```
Command> repschemes;
```

Example 5.11 shows the configuration output from the replication scheme shown in “Propagation scheme” on page 88.

Example 5.11

```
Replication Scheme REPL.PROPAGATOR:

  Element: A
   Type: Table REPL.TAB
   Master Store: CENTRALDS on FINANCE Transmit Durable
   Subscriber Store: PROPDS on NETHANDLER
  Element: B
  Type: Table REPL.TAB
  Propagator Store: PROPDS on NETHANDLER Transmit Durable
  Subscriber Store: BACKUP1DS on BACKUPSYSTEM1
  Subscriber Store: BACKUP2DS on BACKUPSYSTEM2

  Store: BACKUP1DS on BACKUPSYSTEM1
    Port: (auto)
    Log Fail Threshold: (none)
    Retry Timeout: 120 seconds
    Compress Traffic: Disabled

  Store: BACKUP2DS on BACKUPSYSTEM2
    Port: (auto)
    Log Fail Threshold: (none)
    Retry Timeout: 120 seconds
    Compress Traffic: Disabled

  Store: CENTRALDS on FINANCE
    Port: (auto)
    Log Fail Threshold: (none)
    Retry Timeout: 120 seconds
    Compress Traffic: Disabled

  Store: PROPDS on NETHANDLER
    Port: (auto)
    Log Fail Threshold: (none)
    Retry Timeout: 120 seconds
    Compress Traffic: Disabled
```

From the command line: ttRepAdmin -showconfig

To display the configuration of your replicated data stores from the command line, use the **ttRepAdmin** utility with the `-showconfig` option:

```
ttRepAdmin -showconfig -dsn masterDSN
```

Example 5.12 shows the configuration output from the propagated data stores configured by the replication scheme shown in "Propagation scheme" on page 88.

Example 5.12

```
> ttRepAdmin -showconfig -dsn centralds
Self host "finance", port auto, name "centralds", LSN 0/155656,
timeout 120, threshold 0

List of subscribers
-----------------
Peer name         Host name                 Port    State  Proto
---------------- ------------------------ ------ ------- -----
propds            nethandler                Auto    Start     10
```

```
Last Msg Sent Last Msg Recv Latency TPS     RecordsPS Logs
------------- ------------- ------- ------- --------- ----
0:01:12       -               19.41       5        52    2

List of tables and subscriptions
--------------------------------

Table details
-------------
Table : repl.tab       Timestamp updates : -

Master Name               Subscriber Name
-----------               -------------
centralds                 propds

Table details
-------------
Table : repl.tab       Timestamp updates : -

Master Name               Subscriber name
-----------               -------------
propds                    backup1ds
propds                    backup2ds
```

See Example 5.10 on page 125 for the meaning of the “List of subscribers” fields. The “Table details” fields list the table and the names of its master (Sender) and subscriber data stores.

From a program: SQL SELECT statements

To display the configuration of your replicated data stores from a program, use the following SQL SELECT statements to query the TTREP.TTSTORES, TTREP.REPSTORES, TTREP.REPNETWORK, SYS.MONITOR, TTREP.REPELEMENTS, and TTREP.REPSUBSCRIPTIONS tables:

```
SELECT t.host_name, t.rep_port_number, t.tt_store_name,
s.peer_timeout, s.fail_threshold
 FROM ttrep.ttstores t, ttrep.repstores s
  WHERE t.is_local_store = 0X01
   AND t.tt_store_id = s.tt_store_id;
```

```
SELECT t1.tt_store_name, t1.host_name, t1.rep_port_number,
p.state, p.protocol, p.timesend, p.timerecv, p.latency,
p.tps, p.recspersec, t3.last_log_file - p.sendlsnhigh + 1
  FROM ttrep.reppeers p, ttrep.ttstores t1, ttrep.ttstores t2,
   sys.monitor t3
  WHERE p.tt_store_id = t2.tt_store_id
   AND t2.is_local_store = 0X01
   AND p.subscriber_id = t1.tt_store_id
   AND (p.state = 0 OR p.states = 1);
```

```
SELECT ds_obj_owner, DS_OBJ_NAME, t1.tt_store_name,t2.tt_store_name
  FROM ttrep.repelements e, ttrep.repsubscriptions s,
   ttrep.ttstores t1, ttrep.ttstores t2
  WHERE s.element_name = e.element_name
   AND e.master_id = t1.tt_store_id
   AND s.subscriber_id = t2.tt_store_id
  ORDER BY ds_obj_owner, ds_obj_name;
```

Use the **ttBookmark** procedure to obtain the replication hold LSN, as described in "From a program: ttBookMark procedure" on page 131.

Example 5.13 The output from the above queries for the data stores configured by the replication scheme shown in "Propagation scheme" on page 88 might look like the following:

```
< finance, 0, centralds, 120, 0 >
< propds, nethandler, 0, 0, 7, 1004378953, 0, -1.00000000000000, -1, -1, 1 >
< repl, tab, centralds, propds >
< repl, tab, propds, backup1ds >
< repl, tab, propds, backup2ds >
```

See Example 5.6 on page 121 for descriptions for the first three columns in the first row (minus the Replication hold LSN). The fourth column is the TIMEOUT value that defines the amount of time a data store waits for a response from another data store before resending a message. The last column is the log failure threshold value described in "Setting the log failure threshold" on page 108.

See Example 5.10 on page 125 for a description of the second row. The last three rows show the replicated table and the names of its master (sender) and subscriber (receiver) data stores.

Show replicated log records

Transactions are stored in the log in the form of *log records*. You can use *bookmarks* to detect which log records have or have not been replicated by a master data store.

A bookmark consists of *log sequence numbers* (LSNs) that identify the location of particular records in the transaction log that you can use to gauge replication performance. The LSNs associated with a bookmark are: *hold LSN*, *last written LSN*, and *last LSN forced to disk*. The *hold LSN* describes the location of the lowest (or oldest) record held in the log for possible transmission to a subscriber. You can compare the *hold LSN* with the *last written LSN* to determine the number of records in the transaction log that have not yet been transmitted to the subscribers. The *last LSN forced to disk* describes the last records saved in a log file on disk.

A more accurate way to monitor replication to a particular subscriber is to look at the *send LSN* for the subscriber, which consists of the SENDLSNHIGH and SENDLSNLOW fields in the TTREP.REPNETWORK table. In contrast to the *send LSN* value, the *hold LSN* returned in a bookmark is computed every 10 seconds to describe the minimum *send LSN* for all the subscribers, so it provides a more general view of replication progress that does not account for the progress of replication to the individual subscribers. However, due to the asynchronous nature of replication acknowledgements, which was mainly done to improve performance, the *send LSN* can also be some distance behind. But the *send LSN* for a subscriber is the most accurate value available and is always ahead of the *hold LSN*.

You can display replicated log records:

- From the command line: ttRepAdmin -bookmark
- From a program: ttBookMark procedure

From the command line: ttRepAdmin -bookmark

To display the location of the bookmarks from the command line, use the **ttRepAdmin** utility with the `-bookmark` option:

```
> ttRepAdmin -dsn masterds -bookmark
Replication hold LSN ...... 10/927692
Last written LSN .......... 10/928908
Last LSN forced to disk ... 10/280540
```

Each LSN is defined by two values:

Log file number / Offset in log file

The LSNs output from **ttRepAdmin** `-bookmark` are:

Line	Description
Replication hold LSN	The location of the lowest (or oldest) record held in the log for possible transmission to a subscriber. A value of -1/-1 indicates replication is in the **Stop** state with respect to all subscribers (or the queried data store is not a master data store).
Last written LSN	The location of the most recently generated transaction log record for the data store.
Last LSN forced to disk	The location of the most recent transaction log record written to the disk.

From a program: ttBookMark procedure

To display the location of the bookmarks from a program, use the **ttBookmark** procedure.

Example 5.14

```
> ttIsql masterds

Command> call ttBookMark();
< 10, 928908, 10, 280540, 10, 927692 >
1 row found.
```

The first two columns in the returned row define the “Last written LSN,” the next two columns define the “Last LSN forced to disk,” and the last two columns define the “Replication hold LSN.”

Show replication status

You can use the **ttRepAdmin** utility with the `-showstatus` option to display the current status of the replication agent. The status output includes the bookmark locations, port numbers, and communication protocols used by the replication agent for the queried data store.

The output from **ttRepAdmin** `-showstatus` includes the status of the MAIN thread and the TRANSMITTER and RECEIVER threads used by the replication agent. A master data store has a TRANSMITTER thread and a subscriber data store has a RECEIVER thread. A data store that serves a master/subscriber role in a bidirectional replication scheme has both a TRANSMITTER and a RECEIVER thread.

Each replication agent has a single REPLISTENER thread that listens on a port for peer connections. On a master data store, the REPLISTENER thread starts a separate TRANSMITTER thread for each subscriber data store. On a subscriber data store, the REPLISTENER thread starts a separate RECEIVER thread for each connection from a master.

If the TimesTen daemon requests that the replication agent stop or if a fatal error occurs in any of the other threads used by the replication agent, the MAIN thread waits for the other threads to gracefully terminate. The TimesTen daemon may or may not restart the replication agent, depending upon certain fatal errors. The REPLISTENER thread never terminates during the lifetime of the replication agent. A TRANSMITTER or RECEIVER thread may stop but the replication agent may restart it. The RECEIVER thread terminates on errors from which it cannot recover or when the master disconnects.

Example 5.15 shows **ttRepAdmin** `-showstatus` output for a unidirectional replication scheme in which the `rep1` data store is the master and `rep2` data store is the subscriber. The first **ttRepAdmin** `-showstatus` output shows the status of the `rep1` data store and its TRANSMITTER thread. The second output shows the status of the `rep2` data store and its RECEIVER thread.

Following the example are sections that describe the meaning of each field in the **ttRepAdmin** `-showstatus` output:

- MAIN thread status fields
- Replication peer status fields
- TRANSMITTER thread status fields
- RECEIVER thread status fields

Example 5.15 Consider the unidirectional replication scheme from the `rep1` data store to the `rep2` data store:

```
CREATE REPLICATION r
ELEMENT e1 TABLE t
   MASTER rep1
   SUBSCRIBER rep2;
```

The replication status for the `rep1` data store should look similar to the following:

```
> ttRepAdmin -showstatus rep1

DSN                           : rep1
Process ID                    : 1980
Replication Agent Policy      : MANUAL
Host                          : MYHOST
RepListener Port              : 1113 (AUTO)
Last write LSN                : 0.1487928
Last LSN forced to disk       : 0.1487928
Replication hold LSN          : 0.1486640
Replication Peers:
   Name                     : rep2
   Host                     : MYHOST
   Port                     : 1154 (AUTO)
   Replication State        : STARTED
   Communication Protocol   : 12

TRANSMITTER thread(s):
 For                     : rep2
   Start/Restart count   : 2
   Send LSN              : 0.1485960
   Transactions sent     : 3
   Total packets sent    : 10
   Tick packets sent     : 3
   MIN sent packet size  : 48
   MAX sent packet size  : 460
   AVG sent packet size  : 167
   Last packet sent at   : 17:41:05
   Total Packets received: 9
   MIN rcvd packet size  : 48
   MAX rcvd packet size  : 68
   AVG rcvd packet size  : 59
   Last packet rcvd'd at : 17:41:05
   Earlier errors (max 5):
   TT16060 in transmitter.c (line 3590) at 17:40:41 on 08-25-2004
   TT16122 in transmitter.c (line 2424) at 17:40:41 on 08-25-2004
```

The replication status for the rep2 data store should look similar to the following:

```
> ttRepAdmin -showstatus rep2

DSN                           : rep2
Process ID                    : 2192
Replication Agent Policy      : MANUAL
Host                          : MYHOST
RepListener Port              : 1154 (AUTO)
Last write LSN                : 0.416464
Last LSN forced to disk       : 0.416464
Replication hold LSN          : -1.-1

Replication Peers:
   Name                       : rep1
   Host                       : MYHOST
   Port                       : 0 (AUTO)
   Replication State          : STARTED
   Communication Protocol     : 12

RECEIVER thread(s):
 For                       : rep1
   Start/Restart count     : 1
   Transactions received : 0
   Total packets sent      : 20
   Tick packets sent       : 0
   MIN sent packet size    : 48
   MAX sent packet size    : 68
   AVG sent packet size    : 66
   Last packet sent at     : 17:49:51
   Total Packets received: 20
   MIN rcvd packet size    : 48
   MAX rcvd packet size    : 125
   AVG rcvd packet size    : 52
   Last packet rcvd'd at : 17:49:51
```

MAIN thread status fields

The following fields are output for the MAIN thread in the replication agent for the queried data store.

MAIN Thread	Description
DSN	Name of the data store to be queried.
Process ID	Process Id of the replication agent.
Replication Agent Policy	The restart policy, as described in "Starting and stopping the replication agents" on page 111

MAIN Thread	Description
Host	Name of the machine that hosts this data store.
RepListener Port	TCP/IP port used by the replication agent to listen for connections from the TRANSMITTER threads of remote replication agents. A value of 0 indicates that this port has been assigned automatically to the replication agent (the default), rather than being specified as part of a replication scheme.
Last write LSN	The location of the most recently generated transaction log record for the data store. See "Show replicated log records" on page 130 for more information.
Last LSN forced to disk	The location of the most recent transaction log record written to the disk. See "Show replicated log records" on page 130 for more information.
Replication hold LSN	The location of the lowest (or oldest) record held in the log for possible transmission to a subscriber. A value of -1/-1 indicates replication is in the **Stop** state with respect to all subscribers. See "Show replicated log records" on page 130 for more information.

Replication peer status fields

The following fields are output for each replication peer that participates in the replication scheme with the queried data store. A "peer" could play the role of master, subscriber, propagator or both master and subscriber in a bidirectional replication scheme.

Replication Peers	Description
Name	Name of a data store that is a replication peer to this data store.
Host	Host machine of peer data store.
Port	TCP/IP port used by the replication agent for the peer data store. A value of 0 indicates this port has been assigned automatically to the replication agent (the default), rather than being specified as part of a replication scheme.

Replication Peers	Description
Replication State	Current replication state of the replication peer with respect to the queried data store (see "Setting the replication state of subscribers" on page 114 for information).
Communication Protocol	Internal protocol used by replication to communicate between the peers. (For internal use only.)

TRANSMITTER thread status fields

The following fields are output for each TRANSMITTER thread used by a master replication agent to send transaction updates to a subscriber. A master with multiple subscribers has multiple TRANSMITTER threads.

Note: The counts in the TRANSMITTER output begin to accumulate when the replication agent is started. These counters are reset to 0 only when the replication agent is started or restarted.

TRANSMITTER Thread	Description
For	Name of the subscriber data store that is receiving replicated data from this data store.
Start/Restart count	Number of times this TRANSMITTER thread was started or restarted by the replication agent due to a temporary error, such as operation timeout, network failure, and so on.
Send LSN	The last LSN transmitted to this peer. See "Show replicated log records" on page 130 for more information.
Transactions sent	Total number of transactions sent to the subscriber.
Total packets sent	Total number of packets sent to the subscriber (including tick packets)

TRANSMITTER Thread	**Description**
Tick packets sent	Total number of tick packets sent. Tick packets are used to maintain a “heartbeat” between the master and subscriber. You can use this value to determine how many of the 'Total packets sent' packets are not related to replicated data.
MIN sent packet size	Size of the smallest packet sent to the subscriber.
MAX sent packet size	Size of the largest packet sent to the subscriber.
AVG sent packet size	Average size of the packets sent to the subscriber.
Last packet sent at	Time of day last packet was sent (24-hour clock time)
Total packets received	Total packets received from the subscriber (tick packets and acknowledgement data)
MIN rcvd packet size	Size of the smallest packet received
MAX rcvd packet size	Size of the largest packet received
AVG rcvd packet size	Average size of the packets received
Last packet rcvd at	Time of day last packet was received (24-hour clock time)
Earlier errors (max 5)	Last five errors generated by this thread

RECEIVER thread status fields

The following fields are output for each RECEIVER thread used by a subscriber replication agent to receive transaction updates from a master. A subscriber that is updated by multiple masters has multiple RECEIVER threads.

Note: The counts in the RECEIVER output begin to accumulate when the replication agent is started. These counters are reset to 0 only when the replication agent is started or restarted.

RECEIVER Thread	Description
For	Name of the master data store that is sending replicated data from this data store
Start/Restart count	Number of times this RECEIVER thread was started or restarted by the replication agent due to a temporary error, such as operation timeout, network failure, and so on.
Transactions received	Total number of transactions received from the master
Total packets sent	Total number of packets sent to the master (tick packets and acknowledgement data)
Tick packets sent	Total number of tick packets sent to the master. Tick packets are used to maintain a “heartbeat” between the master and subscriber. You can use this value to determine how many of the 'Total packets sent' packets are not related to acknowledgement data.
MIN sent packet size	Size of the smallest packet sent to the master
MAX sent packet size	Size of the largest packet sent to the master
AVG sent packet size	Average size of the packets sent to the master
Last packet sent at	Time of day last packet was sent to the master (24-hour clock time)
Total packets received	Total packets of acknowledgement data received from the master
MIN rcvd packet size	Size of the smallest packet received
MAX rcvd packet size	Size of the largest packet received
AVG rcvd packet size	Average size of the packets received
Last packet rcvd at	Time of day last packet was received (24-hour clock time)

Show the return service status for a subscriber

As described in "When to manually disable return service blocking" on page 67, a replication scheme that makes use of a return service (either RETURN TWOSAFE or RETURN RECEIPT) can be configured with a DISABLE RETURN failure policy to disable return service blocking for unresponsive subscribers.

You can determine whether the return service for particular subscriber has been disabled by the DISABLE RETURN failure policy by calling the **ttRepSyncSubscriberStatus** built-in procedure or by means of the SNMP trap, **ttRepReturnTransitionTrap**. The **ttRepSyncSubscriberStatus** procedure returns a value of '1' to indicate the return service has been disabled for the subscriber, or a value of '0' to indicate that the return service is still enabled.

Example 5.16

To use **ttRepSyncSubscriberStatus** to obtain the return receipt status of the *subscriberds* data store with respect to its master data store, *masterDSN*, enter:

```
> ttIsql masterDSN
```

```
Command> CALL ttRepSyncSubscriberStatus ('subscriberds');
< 0 >
1 row found.
```

This result indicates that the return service is still enabled.

See "DISABLE RETURN" on page 68 for more information.

6

Altering Replication

This chapter describes how to alter an existing replication system. Table 6.1 lists the tasks often performed on an existing replicated system.

Table 6.1 Tasks performed on an existing replicated system

Task	What you do
Alter or drop a replication scheme	See "Altering a replication scheme" on page 141 and "Dropping a replication scheme" on page 150
Alter a table used in a replication scheme	See "Altering a replicated table" on page 149
Truncate a table used in a replication scheme	See "Truncating a replicated table" on page 149
Change the replication state of a subscriber data store	See "Setting the replication state of subscribers" on page 114
Resolve update conflicts	See "Replication conflict detection and resolution" on page 167 in Chapter 8.
Recover from failures	See "Managing data store failover and recovery" on page 183 in Chapter 8.
Upgrade data store	Use the **ttMigrate** and **ttRepAdmin** utilities, as described in Chapter 3, "Data Store Upgrades" in the *Oracle TimesTen In-Memory Database Installation Guide*.

Altering a replication scheme

You can use ALTER REPLICATION to alter your replication scheme on the master and subscriber data stores. Any alterations on the master store must also be made on its subscribers.

Note: If TimesTen was installed with Access Control enabled, you must have DDL privileges to the data store to use the ALTER REPLICATION statement. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

If you use ALTER REPLICATION to change a replication scheme that specifies a DATASTORE element, then:

- You cannot use SET NAME to change the name of the DATASTORE element
- You cannot use SET CHECK CONFLICTS to enable conflict resolution

Most ALTER REPLICATION operations are supported only when the replication agent is stopped (**ttAdmin** `-repStop`). However, it is possible to dynamically add a data store to a replication scheme while the replication agent is running, as described in "Creating and adding a subscriber data store" on page 146.

The procedure for ALTER REPLICATION operations that require the replication agents to be stopped is:

1. Use the **ttRepStop** procedure or **ttAdmin** `-repStop` to stop the replication agent for the master and subscriber data stores. While the replication agents are stopped, changes to the master data store are stored in the log.
2. Issue the same ALTER REPLICATION statement on both master and subscriber data stores.
3. Use the **ttRepStart** procedure or **ttAdmin** `-repStart` to restart the replication agent for the master and subscriber data stores. The changes stored in the master data store log are sent to the subscriber data store.

This section includes the following topics:

- Adding a table or sequence to an existing replication scheme
- Adding a cache group to an existing replication scheme
- Adding a DATASTORE element to an existing replication scheme
- Dropping a table or sequence from a replication scheme
- Creating and adding a subscriber data store
- Dropping a subscriber data store
- Changing a TABLE or SEQUENCE element name
- Replacing a master data store
- Eliminating conflict detection
- Eliminating the return receipt service
- Changing the port number
- Changing the replication route

Adding a table or sequence to an existing replication scheme

There are two ways to add a table or sequence to an existing replication scheme:

- When the element level of the replication scheme is TABLE or SEQUENCE, use the ALTER REPLICATION statement with the ADD ELEMENT clause to add a table or sequence. See Example 6.1.
- When the element level of the replication scheme is DATASTORE, use the ALTER REPLICATION statement with the ALTER ELEMENT clause to include a table or sequence. See Example 6.2.

Example 6.1 This example uses the replication scheme repl.r1, which was defined in Example 3.29 on page 89. It alters replication scheme repl.r1 to add sequence seq and table westleads, which will be updated on data store westds and replicated to data store eastds.

```
ALTER REPLICATION repl.r1
   ADD ELEMENT elem_seq SEQUENCE repl.seq
      MASTER westds ON "westcoast"
      SUBSCRIBER eastds ON "eastcoast"
   ADD ELEMENT elem_westleads TABLE repl.westleads
      MASTER westds ON "westcoast"
      SUBSCRIBER eastds ON "eastcoast";
```

Example 6.2 Add the sequence my.seq and the table my.tab1 to the ds1 DATASTORE element in my.rep1 replication scheme.

```
ALTER REPLICATION my.rep1
  ALTER ELEMENT ds1 DATASTORE
     INCLUDE SEQUENCE my.seq
  ALTER ELEMENT ds1 DATASTORE
     INCLUDE TABLE my.tab1;
```

Adding a cache group to an existing replication scheme

You can add a cache group to a DATASTORE element in an existing replication scheme. Use the ALTER REPLICATION statement with the ALTER ELEMENT clause and the INCLUDE CACHE GROUP clause.

Example 6.3 Add my.cg1 cache group to the ds1 DATASTORE element in my.rep1 replication scheme.

```
ALTER REPLICATION my.rep1
  ALTER ELEMENT ds1 DATASTORE
     INCLUDE CACHE GROUP my.cg1;
```

Adding a DATASTORE element to an existing replication scheme

You can add a DATASTORE element to an existing replication scheme by using the ALTER REPLICATION statement with the ADD ELEMENT clause. All tables except temporary tables, materialized view, and nonmaterialized views are included in the data store if you do not use the INCLUDE or EXCLUDE clauses. See "Including tables, sequences or cache groups when you add a DATASTORE element" on page 144 and "Excluding a table, sequence or cache group when you add a DATASTORE element" on page 144.

Example 6.4 Add a DATASTORE element to an existing replication scheme.

```
ALTER REPLICATION my.rep1
  ADD ELEMENT ds1 DATASTORE
     MASTER rep2
     SUBSCRIBER rep1, rep3;
```

Including tables, sequences or cache groups when you add a DATASTORE element

You can restrict replication to specific tables, sequences and cache groups when you add a data store to an existing replication scheme. Use the ALTER REPLICATION statement with the ADD ELEMENT clause and the INCLUDE TABLE clause, INCLUDE SEQUENCE clause or INCLUDE CACHE GROUP clause. You can have one INCLUDE clause for each object type (table, sequence and cache group) in the same ALTER REPLICATION statement.

Example 6.5 Add the `ds1` DATASTORE element to `my.rep1` replication scheme. Include the table `my.tab2`, the sequence `my.seq`, and the cache groups `my.cg2` and `my.cg3` in the DATASTORE element.

```
ALTER REPLICATION my.rep1
  ADD ELEMENT ds1 DATASTORE
     MASTER rep2
     SUBSCRIBER rep1, rep3
     INCLUDE TABLE my.tab2
     INCLUDE SEQUENCE my.seq
     INCLUDE CACHE GROUP my.cg2, my.cg3;
```

Excluding a table, sequence or cache group when you add a DATASTORE element

You can exclude tables, sequences or cache groups when you add a DATASTORE element to an existing replication scheme. Use the ALTER REPLICATION statement with the ADD ELEMENT clause and the EXCLUDE TABLE clause, EXCLUDE SEQUENCE clause or EXCLUDE CACHE GROUP clause. You can have one EXCLUDE clause for each object type (table, sequence and cache group) in the same ALTER REPLICATION statement.

Example 6.6 Add the ds2 DATASTORE element to a replication scheme, but exclude the table my.tab1, the sequence my.seq and the cache groups my.cg0 and my.cg1.

```
ALTER REPLICATION my.rep1
  ADD ELEMENT ds2 DATASTORE
     MASTER rep2
     SUBSCRIBER rep1
     EXCLUDE TABLE my.tab1
     EXCLUDE SEQUENCE my.seq
     EXCLUDE CACHE GROUP my.cg0, my.cg1;
```

Dropping a table or sequence from a replication scheme

This section includes the following topics:

- Dropping a table or sequence that is replicated as part of a DATASTORE element
- Dropping a table or sequence that is replicated as a TABLE or SEQUENCE element

Dropping a table or sequence that is replicated as part of a DATASTORE element

To drop a table or sequence that is part of a replication scheme at the DATASTORE level, complete the following tasks:

1. Stop the replication agent.
2. Exclude the table or sequence from the DATASTORE element in the replication scheme.
3. Drop the table or sequence.

If you have more than one DATASTORE element that contains the table or sequence, then you must exclude the table or sequence from each element before you drop it.

Example 6.7 Exclude the table my.tab1 from the ds1 DATASTORE element in the my.rep1 replication scheme. Then drop the table.

```
ALTER REPLICATION my.rep1
  ALTER ELEMENT ds1 DATASTORE
     EXCLUDE TABLE my.tab1;
DROP TABLE my.tab1;
```

Dropping a table or sequence that is replicated as a TABLE or SEQUENCE element

To drop a table that is part of a replication scheme at the TABLE or SEQUENCE level, complete the following tasks:

1. Stop the replication agent.
2. Drop the element from the replication scheme.
3. Drop the table or sequence.

Example 6.8 Drop the SEQUENCE element `elem_seq` from the replication scheme `repl.r1`. Then drop the sequence `repl.seq`.

```
ALTER REPLICATION repl.r1
  DROP ELEMENT elem_seq;
DROP SEQUENCE repl.seq;
```

Creating and adding a subscriber data store

You can add a new subscriber data store while the replication agents are running. To add a data store to a replication scheme, do the following:

1. Make sure the new data store does not exist.
2. Apply the appropriate statements to all participating data stores:

```
ALTER REPLICATION ...
     ALTER ELEMENT ...
         ADD SUBSCRIBER ...
```

3. Run the **ttRepAdmin** `-duplicate` command to copy the contents of the master data store to the newly created subscriber. You can use the `-setMasterRepStart` option to ensure that any updates made to the master after the duplicate operation has started are also copied to the subscriber.
4. Start the replication agent on the newly created data store (**ttAdmin** `-repStart`).

Example 6.9 This example alters the `repl.r1` replication scheme to add an additional subscriber (`backup3`) to the `westleads` table (step 2. above):

```
ALTER REPLICATION repl.r1
  ALTER ELEMENT elem_westleads
      ADD SUBSCRIBER backup3 ON "backupserver";
```

Dropping a subscriber data store

Stop the replication agent before you drop a subscriber data store.

This example alters the `repl.r1` replication scheme to drop the `backup3` subscriber for the `westleads` table:

Example 6.10

```
ALTER REPLICATION repl.r1
   ALTER ELEMENT elem_westleads
      DROP SUBSCRIBER backup3 ON "backupserver";
```

Changing a TABLE or SEQUENCE element name

Stop the replication agent before you change a TABLE or SEQUENCE element name.

Change the element name of the `westleads` table from `elem_westleads` to `newelname`:

Example 6.11

```
ALTER REPLICATION repl.r1
   ALTER ELEMENT Eelem_westleads
      SET NAME newelname;
```

Note: You cannot use the SET NAME clause to change the name of a DATASTORE element.

Replacing a master data store

Stop the replication agent before you replace a master data store.

In this example, `newwestds` is made the new master for all elements currently configured for the master, `westds`:

Example 6.12

```
ALTER REPLICATION repl.r1
   ALTER ELEMENT * IN westds
      SET MASTER newwestds;
```

Eliminating conflict detection

In this example, conflict detection configured by the CHECK CONFLICTS clause in the scheme shown in Example 8.2 on page 172 is eliminated for the `elem_accounts_1` table:

Example 6.13

```
ALTER REPLICATION repl.r1
   ALTER ELEMENT elem_accounts_1
      SET NO CHECK;
```

See "Replication conflict detection and resolution" on page 167 for a detailed discussion on conflict checking.

Eliminating the return receipt service

In this example, the return receipt service is eliminated for the first subscriber in the scheme shown in Example 3.29 on page 89:

Example 6.14

```
ALTER REPLICATION repl.r1
   ALTER ELEMENT elem_waccounts
      ALTER SUBSCRIBER eastds ON "eastcoast"
         SET NO RETURN;
```

Changing the port number

The *port number* is the TCP/IP port number on which a subscribing data store's replication agent accepts connection requests from its master replication agent. See "Dynamic vs. static port assignments" on page 55 for details on how to assign port to the replication agents.

In this example, the `repl.r1` replication scheme is altered to change the `eastds` data store's port number to 22251:

Example 6.15

```
ALTER REPLICATION repl.r1
   ALTER STORE eastds ON "eastcoast"
     SET PORT 22251;
```

Changing the replication route

If a replication host has multiple network interfaces, you may specify which interfaces are used for replication traffic using the ROUTE clause. If you need to change which interfaces are used by replication, you may do so by dropping and adding IP addresses from or to a ROUTE clause.

Example 6.16

In this example, the `rep.r1` replication scheme is altered to change the priority 2 IP address for the master data store from 192.168.1.100 to 192.168.1.101:

```
ALTER REPLICATION repl.r1
   DROP ROUTE MASTER eastds ON "eastcoast"
              SUBSCRIBER westds ON "westcoast”
              MASTERIP "192.168.1.100"
   ADD ROUTE MASTER eastds ON "eastcoast"
             SUBSCRIBER westds ON "westcoast"
             MASTERIP "192.168.1.101" PRIORITY 2;
```

Altering a replicated table

You can use ALTER TABLE to add or drop columns on the master data store. The ALTER TABLE operation is replicated to alter the subscriber data stores.

If you use ALTER TABLE on a data store configured for bidirectional replication, first stop updates to the table on all of the replicated data stores and confirm all replicated updates to the table have been received by the data stores before issuing the ALTER TABLE statement. Do not resume updates until the ALTER TABLE operation has been replicated to all data stores. This is necessary to ensure there will be no write operations in the pre-altered format after the table is altered on all data stores.

Note: You can use the **ttRepSubscriberWait** procedure or monitoring tools described in Chapter 5, "Monitoring Replication" to confirm the updates have been received and committed on the data stores.

Also, if you are executing a number of successive ALTER TABLE operations on a data store, you should only proceed with the next ALTER TABLE after you have confirmed the previous ALTER TABLE has reached all of the subscribers.

Note: You can use the ALTER TABLE statement to change default column values, but the ALTER TABLE statement is not replicated. Thus default column values need not be identical on all nodes.

Truncating a replicated table

You can use TRUNCATE TABLE to delete all of the rows of a table without dropping the table itself. Truncating a table is faster than using a DELETE FROM `table` statement.

Truncate operations on replicated tables are replicated and result in truncating the table on the subscriber data store. Unlike delete operations, however, the individual rows are not deleted. Even if the contents of the tables do not match at the time of the truncate operation, the rows on the subscriber data store are deleted anyway.

The TRUNCATE statement replicates to the subscriber, even when no rows are operated upon.

When tables are being replicated with timestamp conflict checking enabled, conflicts are not reported.

Dropping a replication scheme

You can use the DROP REPLICATION statement to remove a replication scheme from a data store. You cannot drop a replication scheme when master catchup is required unless it is the only replication scheme in the data store.

Note: If TimesTen was installed with Access Control enabled, you must have DDL privileges to the data store to use the DROP REPLICATION statement. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

You must stop the replication agent before you drop a replication scheme.

Example 6.17

To remove the `repl.repscheme` replication scheme from a data store, enter the following:

```
DROP REPLICATION repl.repscheme;
```

If you are dropping replicated tables, you must drop the replication scheme *before* dropping the replicated tables. Otherwise, you receive an error indicating that you have attempted to drop a replicated table or index.

Example 6.18

To remove the `repl.tab` table and `repl.repscheme` replication scheme from a data store, enter the following:

```
DROP REPLICATION repl.repscheme;
DROP TABLE repl.tab;
```

7

Administering an Active Standby Pair

This chapter describes how to administer an active standby pair. It includes the following topics:

See "Active standby pair with read-only subscribers" on page 24 for an overview of active standby pairs.

Restrictions on active standby pairs

When you are planning an active standby pair, keep in mind the following restrictions:

- You can specify at most 127 subscriber data stores.
- The active master data store and the standby master data store should be on the same LAN.
- The clock skew between the active node and the standby node cannot exceed 250 milliseconds.
- The DATASTORE element is required.
- For the initial set-up, you can create a standby master data store only by duplicating the active master data store with the **ttRepAdmin** `-duplicate` utility or the **ttRepDuplicateEx** C function.
- Read-only subscribers can be created only by duplicating the standby master data store. If the standby master data store is unavailable, then the read-only subscribers can be created by duplicating the active master standby store.

- After failover, the new standby master data store can only be recovered from the active master data store by duplicating the active master data store *unless* return twosafe replication is used between the active and the standby master data stores. If return twosafe replication is used, the automated master catch-up feature may be used instead. See "Automatic catch-up of a failed master data store" on page 185.
- Operations on replicated tables are not allowed on the standby master data store and the subscriber stores. However, operations on sequences and XLA bookmarks *are* allowed on the standby master data store and the subscriber stores.
- Replication to the subscribers can occur only in asynchronous mode.
- ALTER REPLICATION statements can be executed only on the active master data store. If ALTER REPLICATION is executed on the active master data store, then the standby master data store must be regenerated by duplicating the active master data store. All subscribers must also be regenerated from the standby master data store. See "Reversing the roles of the active and standby master data stores" on page 161.

The following restrictions apply to active standby pairs and cache groups:.

- An active standby pair that uses the RETURN TWOSAFE return service cannot contain a synchronous writethrough (SWT) cache group.
- An active standby pair that uses the RETURN TWOSAFE return service cannot contain a USERMANAGED cache group with the PROPAGATE option.
- The replication agent for a master data store in an active standby pair must be stopped before you can create an AWT cache group in that data store.

Master data store states

The master data stores can be in one of the following states:

- ACTIVE - A store in this state is the active master data store. Applications can update its replicated tables.
- STANDBY - A store in this state is the standby master data store. Applications can update only its nonreplicated tables (tables that have been excluded from the replication scheme by using the EXCLUDE TABLE or EXCLUDE CACHE GROUP clauses of the CREATE REPLICATION statement).
- FAILED - A data store in this state is a failed master data store. No updates can be replicated to it.
- IDLE - A store in this state has not yet had its role in the active standby pair assigned. It cannot be updated. Every store comes up in the IDLE state.
- RECOVERING - When a previously failed master data store is synchronizing updates with the active master data store, it is in the RECOVERING state.

You can use the **ttRepStateGet** procedure to discover the state of a master data store.

Active standby pairs with cache groups

An active standby pair configured with an autorefreshed READONLY cache group or an ASYNCHRONOUS WRITETHROUGH cache group can change the role of a data store's cache group automatically as part of failover and recovery, to help ensure high availability of cache instances with minimal chance for data loss.

READONLY cache groups with AUTOREFRESH in an active standby pair

In a READONLY cache group configuration with AUTOREFRESH, the cache group on the active master data store is autorefreshed from the Oracle database and replicates the updates to the standby master, where AUTOREFRESH is also configured on the cache group but is in the PAUSED state. In the event of a failure of the active master, TimesTen automatically reconfigures the standby master to be autorefreshed when it takes over for the failed master data store by setting the AUTOREFRESH STATE to ON.

TimesTen also tracks whether updates that have been autorefreshed from the Oracle database to the active master data store have been replicated to the standby master. This ensures that the autorefresh process picks up from the correct point after the active master fails, and no autorefreshed updates are lost.

This configuration may also include read-only subscriber data stores, which are updated by the standby master data store. This allows the read workload to be distributed across many data stores.

ASYNCHRONOUS WRITETHROUGH cache groups in an active standby pair

An ASYNCHRONOUS WRITETHROUGH (AWT) cache group can be configured as part of an active standby pair in order to ensure high availability and to distribute the application workload. Application updates are made to the active master data store, the updates are replicated to the standby master data store, and then the updates are asynchronously written to the Oracle database by the standby master. At the same time, the updates are also replicated from the standby master to the read-only subscribers, which may be used to distribute the load from reading applications.

When there is no standby master data store, the active master both accepts application updates *and* writes the updates asynchronously to the Oracle database. This situation can occur when the standby master has not yet been created, or when the active master fails and the standby master becomes the new active master. TimesTen automatically performs the necessary reconfiguration of the AWT cache group when the standby master becomes the new active master.

Because updates are always written to both master data stores before being transferred to the Oracle database, neither master data store can ever fall behind the Oracle database in the event of that one of the data stores fails.

Setting up an active standby pair

To set up an active standby pair, complete the following tasks. If you intend to replicate a READONLY cache group with AUTOREFRESH or an ASYNCHRONOUS WRITETHROUGH (AWT) cache group, you must also complete the tasks indicated in *italics*:

1. Create a data store.
2. *If you are replicating an AUTOREFRESH or AWT cache group, set the cache agent user ID and password using* **ttCacheUidPwdSet** *(see* “Defining a Cache Group” on page 31 *of the TimesTen Cache Connect to Oracle Guide) and then start the cache agent for the data store using the* **ttCacheStart** *procedure or the* **ttAdmin** `-cacheStart` *utility (see* “Starting and stopping the cache agent” on page 82 *of the TimesTen Cache Connect to Oracle Guide). Next, create the cache group using the* CREATE CACHE GROUP *statement. If you have created an AUTOREFRESH cache group, the AUTOREFRESH STATE must be set to PAUSED before continuing with the next step.*
3. Create the replication scheme using the CREATE ACTIVE STANDBY PAIR statement.

   In , `master1` and `master2` are designated as the master data stores. `sub3` and `sub4` are designated as the subscriber data stores. The data stores reside on `node1`, `node2`, `node3`, and `node4`. The replication mode is RETURN RECEIPT.

   The *FullStoreName* (`master1 ON "node1"`) is an optional form. It is also correct to specify `master1` alone as the data store name. For example:

   ```
   CREATE ACTIVE STANDBY PAIR master1 ON "node1", master2 ON "node2"
     RETURN RECEIPT
     SUBSCRIBER sub1 ON "node3", sub2 ON "node4"
     STORE master1 ON "node1" PORT 21000 TIMEOUT 30
     STORE master2 ON "node2" PORT 20000 TIMEOUT 30;
   ```

4. Set up the replication agent policy for `master1` and start the replication agent. See “Starting and stopping the replication agents” on page 111.
5. Execute **ttRepStateSet**`('ACTIVE')` on the active master data store (`master1`).
6. *If you are replicating an AUTOREFRESH cache group, load the cache group using the LOAD CACHE GROUP command to begin the autorefresh process. You may also load the cache group if you are replicating an AWT cache group,*

although it is not required. See “LOAD CACHE GROUP” on page 322 *of the* *Oracle TimesTen In-Memory Database SQL Reference Guide* *for more information.*

7. Duplicate the active master data store (`master1`) to the standby master data store (`master2`). You can use either the **ttRepAdmin** `-duplicate` utility or the **ttRepDuplicateEx** C function to duplicate a data store. *If you are replicating an AUTOREFRESH or AWT cache group, you must use the* `-keepCG` *command line option with* **ttRepAdmin** *in order to preserve the cache group.*

8. Set up the replication agent policy on `master2` and start the replication agent. See “Starting and stopping the replication agents” on page 111.

9. Wait for `master2` to enter the STANDBY state. Use the **ttRepStateGet** procedure to check the state of `master2`.

10. *If you are replicating an AUTOREFRESH or AWT cache group, start the cache agent for* `master2` *using the* **ttCacheStart** *procedure or the* **ttAdmin** `-cacheStart` *utility.*

11. Duplicate all of the subscribers (`sub1` and `sub2`) from the standby master data store (`master2`). See “Copying a master data store to a subscriber” on page 105. *If you are replicating an AUTOREFRESH or AWT cache group, you must use the* `-noKeepCG` *command line option with* **ttRepAdmin** *in order to convert the cache group to normal TimesTen tables on the subscribers.*

12. Set up the replication agent policy on the subscribers and start the agent on each of the subscriber stores. See “Starting and stopping the replication agents” on page 111.

Recovering from a failure of the active master data store

This section includes the following topics:

- Recovering when the standby master data store is ready
- Recovering when the standby master data store is not ready
- Failing back to the original nodes

Recovering when the standby master data store is ready

This section describes how to recover the active master data store when the standby master data store is available and synchronized with the active master data store. It includes the following topics:

- When replication is return receipt or asynchronous
- When replication is return twosafe

When replication is return receipt or asynchronous

Complete the following tasks:

1. Stop the replication agent on the failed data store if it has not already been stopped.

2. On the standby master data store, execute **ttRepStateSet**('ACTIVE'). This changes the role of the data store from STANDBY to ACTIVE. *If you are replicating an AUTOREFRESH cache group, this action automatically causes the AUTOREFRESH state to change from PAUSED to ON for this data store.*

3. On the new active master data store, execute **ttRepStateSave**('FAILED', '*failed_store*', '*host_name*'), where *failed_store* is the former active master data store that failed. This step is necessary for the new active master data store to replicate directly to the subscriber data stores.

4. *If you are replicating an AUTOREFRESH or AWT cache group, stop the cache agent on the failed data store if it is not already stopped.*

5. Destroy the failed data store.

6. Duplicate the new active master data store to the new standby master data store. You can use either the **ttRepAdmin** -duplicate utility or the **ttRepDuplicateEx** C function to duplicate a data store. *If you are replicating an AUTOREFRESH or AWT cache group, you must use the* -keepCG -recoveringNode *command line options with* **ttRepAdmin** *in order to preserve the cache group.*

7. Set up the replication agent policy on the new standby master data store and start the replication agent. See "Starting and stopping the replication agents" on page 111.

8. *If you are replicating an AUTOREFRESH or AWT cache group, start the cache agent on the new standby master data store.*

   The standby master data store contacts the active master data store. The active master data store stops sending updates to the subscribers. When the standby master data store is fully synchronized with the active master data store, then the standby master data store enters the STANDBY state and starts sending updates to the subscribers. If you are replicating an AWT cache group, the new standby master data store takes over processing of the cache group automatically when it enters the STANDBY state.

Note: You can verify that the standby master data has entered the STANDBY state by using the **ttRepStateGet** procedure.

When replication is return twosafe

Complete the following tasks:

1. On the standby master data store, execute **ttRepStateSet**`('ACTIVE')`. This changes the role of the data store from STANDBY to ACTIVE. *If you are replicating an AUTOREFRESH cache group, this action automatically causes the AUTOREFRESH state to change from PAUSED to ON for this data store.*
2. On the new active master data store, execute **ttRepStateSave**`('FAILED', '`*`failed_store`*`', '`*`host_name`*`')`, where *`failed_store`* is the former active master data store that failed. This step is necessary for the new active master data store to replicate directly to the subscriber data stores.
3. Connect to the failed data store. This triggers recovery from the local transaction logs. If data store recovery fails, you must continue from Step 5 of the procedure for recovering when replication is return receipt or asynchronous. See "When replication is return receipt or asynchronous" on page 155. *If you are replicating an AUTOREFRESH cache group, the autorefresh state is automatically set to PAUSED.*
4. Verify that the replication agent for the failed data store has restarted. If it has not restarted, then start the replication agent. See "Starting and stopping the replication agents" on page 111.
5. *If you are replicating an AUTOREFRESH or AWT cache group, verify that the cache agent for the failed data store has restarted. If it has not restarted, then start the cache agent. See* "Starting and stopping the cache agent" on page 82 *of the TimesTen Cache Connect to Oracle Guide.*

   When the active master data store determines that it is fully synchronized with the standby master data store, then the standby master store enters the STANDBY state and starts sending updates to the subscribers. If you are replicating an AWT cache group, the new standby master data store takes over processing of the cache group automatically when it enters the STANDBY state.

Note: You can verify that the standby master data has entered the STANDBY state by using the **ttRepStateSet** procedure.

Recovering when the standby master data store is not ready

Consider the following scenarios:

- The standby master data store fails. The active master data store fails before the standby comes back up or before the standby has been synchronized with the active master data store.
- The active master data store fails. The standby master data store becomes ACTIVE, and the rest of the recovery process begins. (See "Recovering from

a failure of the active master data store" on page 155.) The new active master data store fails before the new standby master data store is fully synchronized with it.

In both scenarios, the subscribers may have had more changes applied than the standby master data store.

When the active master data store fails and the standby master data store has not applied all of the changes that were last sent from the active master data store, there are two choices for recovery:

- Recover the *active* master data store from the local transaction logs.
- Recover the *standby* master data store from the local transaction logs.

The choice depends on which data store is available and which is more up to date.

Recover the active master data store

1. Connect to the failed active data store. This triggers recovery from the local transaction logs. *If you are replicating an AUTOREFRESH cache group, the autorefresh state is automatically set to PAUSED.*
2. Verify that the replication agent for the failed active data store has restarted. If it has not restarted, then start the replication agent. See "Starting and stopping the replication agents" on page 111.
3. Execute **ttRepSyncSet**`('ACTIVE')` on the newly recovered store. *If you are replicating an AUTOREFRESH cache group, this action automatically causes the AUTOREFRESH state to change from PAUSED to ON for this data store.*
4. *If you are replicating an AUTOREFRESH or AWT cache group, verify that the cache agent for the failed data store has restarted. If it has not restarted, then start the cache agent. See* "Starting and stopping the cache agent" on page 82 *of the TimesTen Cache Connect to Oracle Guide.*
5. Continue with Step 7. in "Setting up an active standby pair" on page 154.

Recover the standby master data store

1. Connect to the failed standby data store. This triggers recovery from the local transaction logs. *If you are replicating an AUTOREFRESH cache group, the autorefresh state is automatically set to PAUSED.*
2. If the replication agent for the standby data store has automatically restarted, you must stop the replication agent. See "Starting and stopping the replication agents" on page 111.
3. *If you are replicating an AUTOREFRESH or AWT cache group and the cache agent has automatically restarted, you must stop the cache agent. See* "Starting and stopping the cache agent" on page 82 *of the TimesTen Cache Connect to Oracle Guide.*

4. Drop the replication configuration using the DROP ACTIVE STANDBY PAIR statement.
5. *If you are replicating an AWT cache group, drop and recreate all AWT cache groups using the* DROP CACHE GROUP *and* CREATE CACHE GROUP *statements.*
6. Re-create the replication configuration using the CREATE ACTIVE STANDBY PAIR statement.
7. Set up the replication agent policy and start the replication agent. See "Starting and stopping the replication agents" on page 111.
8. Execute **ttRepStateSet**('ACTIVE') on the master data store, giving it the ACTIVE role. *If you are replicating an AUTOREFRESH cache group, this action automatically causes the AUTOREFRESH state to change from PAUSED to ON for this data store.*
9. *If you are replicating an AUTOREFRESH or AWT cache group, start the cache agent. See* "Starting and stopping the cache agent" on page 82 *of the TimesTen Cache Connect to Oracle Guide.*
10. Continue from Step 7. in "Setting up an active standby pair" on page 154.

Failing back to the original nodes

After a successful failover, you may wish to fail back so that the active master data store and the standby master data store are on their original nodes. See "Reversing the roles of the active and standby master data stores" on page 161 for instructions.

Recovering from a failure of the standby master data store

To recover from a failure of the standby master data store, complete the following tasks:

1. Detect the standby master data store failure.
2. If return twosafe service is enabled, the failure of the standby master data store may prevent a transaction in progress from being committed on the active master data store, resulting in error 8170, "Receipt or commit acknowledgement not returned in the specified timeout interval". If so, then call the **ttRepStateSet** procedure with a localAction parameter of 2 (COMMIT) and commit the transaction again. For example:

```
call ttRepSyncSet( null, null, 2);
commit;
```

3. Execute **ttRepStateSave**('FAILED', '*standby_store*', '*host_name*') on the active master data store. After this, as long as the standby data store is unavailable, updates to the active data store are replicated directly to the

subscriber data stores. Subscriber stores may also be duplicated directly from the active master.

4. If the replication agent for the standby data store has automatically restarted, stop the replication agent. See "Starting and stopping the replication agents" on page 111.

5. *If you are replicating an AUTOREFRESH or AWT cache group and the cache agent has automatically restarted, stop the cache agent. See "Starting and stopping the cache agent" on page 82 of the TimesTen Cache Connect to Oracle Guide.*

6. Recover the standby master data store in one of the following ways:

   - Connect to the standby master data store. This triggers recovery from the local transaction logs.
   - Duplicate the standby master data store from the active master data store. You can use either the **ttRepAdmin** `-duplicate` utility or the **ttRepDuplicateEx** C function to duplicate a data store. *If you are replicating an AUTOREFRESH or AWT cache group, you must use the* `-keepCG -recoveringNode` *command line options with* **ttRepAdmin** *in order to preserve the cache group.*

   The amount of time that the standby master data store has been down and the amount of transaction logs that need to be applied from the active master data store determine the method of recovery that you should use.

7. Set up the replication agent policy and start the replication agent. See "Starting and stopping the replication agents" on page 111.

8. *If you are replicating an AUTOREFRESH or AWT cache group, start the cache agent. See "Starting and stopping the cache agent" on page 82 of the TimesTen Cache Connect to Oracle Guide.*

   The standby master data store enters the STANDBY state after the active master data store determines that the two master data stores have been synchronized.

Note: You can verify that the standby master data has entered the STANDBY state by using the **ttRepStateGet** procedure.

Recovering from the failure of a subscriber data store

If a subscriber data store fails, then you can recover it by one of the following methods:

- Connect to the failed subscriber. This triggers recovery from the local transaction logs. Start the replication agent and let the subscriber catch up.
- Duplicate the subscriber from the standby master data store. You can use either the **ttRepAdmin** `-duplicate` utility or the **ttRepDuplicateEx** C function to duplicate a data store. *If you are replicating an AUTOREFRESH*

or AWT cache group, you must use the `-noKeepCG` *command line option with* **ttRepAdmin** *in order to convert the cache group to normal TimesTen tables on the subscriber.*

If the standby master data store is down or in recovery, then duplicate the subscriber from the active master data store.

After the subscriber data store has been recovered, then set up the replication agent policy and start the replication agent. See "Starting and stopping the replication agents" on page 111.

Reversing the roles of the active and standby master data stores

To change the active master data store's role to that of a standby master data store and vice versa:

1. Pause any applications that are generating updates on the current active master data store.
2. Execute **ttRepSubscriberWait** on the active master data store, with the DSN and host of the current standby data store as input parameters. This ensures that all updates have been transmitted to the current standby master data store.
3. Stop the replication agent on the current active master data store. See "Starting and stopping the replication agents" on page 111.
4. *If you are replicating an AUTOREFRESH or AWT cache group, stop the cache agent on the active master data store. See* "Starting and stopping the cache agent" on page 82 *of the* *TimesTen Cache Connect to Oracle Guide*.
5. Execute **ttRepDeactivate** on the current active master data store. This puts the store in the IDLE state. *If you are replicating an AUTOREFRESH cache group, this action automatically causes the AUTOREFRESH state to change from ON to PAUSE for this data store.*
6. Execute **ttRepStateSet**`('ACTIVE')` on the current standby master data store. This store now acts as the active master data store in the active standby pair. *If you are replicating an AUTOREFRESH cache group, this automatically causes the AUTOREFRESH state to change from PAUSED to ON for this data store.*
7. Configure the replication agent policy as needed and start the replication agent on the old active master data store. Use the **ttRepStateGet** procedure to determine when the data store's state has changed from IDLE to STANDBY. The data store now acts as the standby master data store in the active standby pair.
8. *If you are replicating an AUTOREFRESH or AWT cache group, start the cache agent on the old active master data store. See* "Starting and stopping the cache agent" on page 82 *of the* *TimesTen Cache Connect to Oracle Guide.*
9. Resume any applications that were paused in Step 1.

Changing the configuration of an active standby pair

You can change an active standby pair by:

- Adding or dropping a subscriber data store
- Altering store attributes - Only the PORT and TIMEOUT attributes can be set for subscribers. The RELEASE clause cannot be set for any data store in an active standby pair.
- Including tables or cache groups in the active standby pair
- Excluding tables or cache groups from the active standby pair

Make these changes on the active master data store. After you have changed the replication scheme on the active master data store, it no longer replicates updates to the standby master data store or to the subscribers. You must re-create the standby master data store and the subscribers and restart the replication agents.

Use the ALTER ACTIVE STANDBY PAIR statement to change the active standby pair.

To change an active standby pair, complete the following tasks:

1. Stop the replication agent on the active master data store. See “Starting and stopping the replication agents” on page 111
2. *If you are replicating an AUTOREFRESH or AWT cache group, stop the cache agent on the active master data store. See* “Starting and stopping the cache agent” on page 82 *of the TimesTen Cache Connect to Oracle Guide*.
3. Use the ALTER ACTIVE STANDBY PAIR statement to make changes to the replication scheme.
4. Start the replication agent on the active master data store. See “Starting and stopping the replication agents” on page 111
5. *If you are replicating an AUTOREFRESH or AWT cache group, start the cache agent on the active master data store. See* “Starting and stopping the cache agent” on page 82 *of the TimesTen Cache Connect to Oracle Guide*.
6. Destroy the standby master data store and the subscribers.
7. Continue from Step 7. of “Setting up an active standby pair” on page 154. This step describes duplicating the active master data store to the standby master data store.

Example 7.1 Add a subscriber data store to the active standby pair.

```
ALTER ACTIVE STANDBY PAIR
ADD SUBSCRIBER sub1;
```

Example 7.2 Drop subscriber data stores from the active standby pair.

```
ALTER ACTIVE STANDBY PAIR
DROP SUBSCRIBER sub1
DROP SUBSCRIBER sub2;
```

Example 7.3 Alter the PORT and TIMEOUT settings for subscribers `rep3` and `rep4`.

```
ALTER ACTIVE STANDBY PAIR
ALTER STORE sub1 SET PORT 23000 TIMEOUT 180
ALTER STORE sub2 SET PORT 23000 TIMEOUT 180;
```

Example 7.4 Add two tables and a cache group to the active standby pair.

```
ALTER ACTIVE STANDBY PAIR
INCLUDE TABLE tab1, tab2
INCLUDE CACHE GROUP cg0;
```

Upgrading the data stores in an active standby pair

This section includes the following topics:

- Upgrades for TimesTen patch releases on the standby master data store and subscriber stores
- Upgrades for TimesTen patch releases on the active master data store
- Upgrades for major TimesTen releases, application software and hardware

Upgrades for TimesTen patch releases on the standby master data store and subscriber stores

To upgrade to a TimesTen patch release on the standby master data store and subscriber stores, complete the following tasks on each store:

1. Stop the replication agent on the store. See "Starting and stopping the replication agents" on page 111.
2. *If you are upgrading the standby master data store and are replicating an AUTOREFRESH or AWT cache group, stop the cache agent on the data store. See* "Starting and stopping the cache agent" on page 82 *of the TimesTen Cache Connect to Oracle Guide*.
3. Install the TimesTen patch. See Chapter 3, "Performing an offline upgrade" in *Oracle TimesTen In-Memory Database Installation Guide.*
4. *If you are upgrading the standby master data store and are replicating an AUTOREFRESH or AWT cache group, restart the cache agent on the data store. See* "Starting and stopping the cache agent" on page 82 *of the TimesTen Cache Connect to Oracle Guide*.
5. Restart the replication agent.

Upgrades for TimesTen patch releases on the active master data store

Complete the following tasks:

1. Reverse the roles of the active and standby master data stores. See "Reversing the roles of the active and standby master data stores" on page 161.
2. Upgrade the former active master data store, which is now the standby master data store. See "Upgrades for TimesTen patch releases on the standby master data store and subscriber stores" on page 164.
3. If you wish to make the newly upgraded data store the active master data store again, repeat the steps for reversing the roles of the active and standby master data stores. See "Reversing the roles of the active and standby master data stores" on page 161.

Upgrades for major TimesTen releases, application software and hardware

Begin major upgrades on the node with the standby master data store. While this node is being upgraded, there is no standby master data store, and updates on the active master data store are propagated directly to the subscriber stores.

1. Execute **ttRepStateSave**`('FAILED',` *`standby_store`*`,` *`host_name`*`)` from the active master data store.
2. Upgrade the node where the standby master data store resides. See Chapter 3, "Performing an offline upgrade" in *Oracle TimesTen In-Memory Database Installation Guide.*
3. Set the replication policy for the standby master data store and start the replication agent. See "Starting and stopping the replication agents" on page 111.

   When the upgraded standby master data store has become synchronized with the active master data store, the upgraded standby master data store moves from the RECOVERING state to the STANDBY state. The upgraded standby master data store also starts sending updates to the subscribers.
4. Stop the replication agent on the active master data store.
5. On the standby master data store, execute **ttRepStateSet**`('ACTIVE')`. This changes the role of the data store from STANDBY to ACTIVE.
6. On the new active master data store, execute **ttRepStateSave**`('FAILED',` `'`*`upgrade_store`*`',` `'`*`host_name`*`')`, where *`upgrade_store`* is the former active master data store on the node that you are upgrading. This step is necessary for the new active master data store to replicate directly to the subscriber data stores.
7. Destroy the former active master data store.
8. Perform the upgrade on the node where the master data store was destroyed.
9. Duplicate the new standby master data store from the active master data store. You can use either the **ttRepAdmin** `-duplicate` utility or the **ttRepDuplicateEx** C function to duplicate a data store.
10. Start the replication agent on the new standby master data store.
11. Stop the replication agent on the first subscriber data store.
12. Destroy the subscriber data store.
13. Perform the upgrade on the node where the subscriber data store was destroyed.
14. Duplicate the subscriber data store from the standby master data store.
15. Start the replication agent for the duplicated subscriber data store.
16. Repeat steps 11 through 15 for each other subscriber data store.

8

Conflict Resolution and Failure Recovery

This chapter describes:

- Replication conflict detection and resolution
- Managing data store failover and recovery

Replication conflict detection and resolution

Tables in data stores configured in a bidirectional replication scheme, as described in "General workload configuration" on page 20, may be subject to *replication conflicts*. A replication conflict occurs when applications on bidirectionally replicated data stores initiate an UPDATE, INSERT or DELETE operation on the same data item at the same time. If no special steps are taken, each data store can end up in disagreement with the last update made by the other data store.

Three different types of replication conflicts can occur:

- Update conflicts: This type of conflict occurs when concurrently running transactions at different stores make simultaneous UPDATE requests on the same row in the same table, and install different values for one or more columns.
- Uniqueness conflicts: This type of conflict occurs when concurrently running transactions at different stores make simultaneous INSERT requests for a row in the same table that has the same primary or unique key, but different values for one or more other columns.
- Delete conflicts: This type of conflict occurs when a transaction at one store deletes a row while a concurrent transaction at another store simultaneously updates or inserts the same row. Currently, TimesTen can detect delete/update conflicts, but cannot detect delete/insert conflicts. TimesTen cannot resolve either type of delete conflict.

See "Conflict reporting" on page 174 for example reports generated by TimesTen upon detecting update, uniqueness, and delete conflicts.

Note: TimesTen does not detect conflicts involving TRUNCATE TABLE statements.

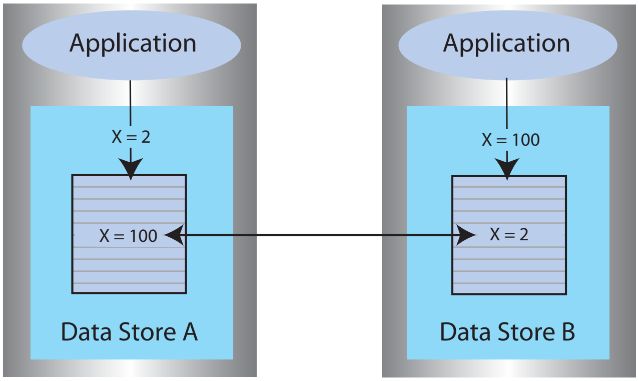
Update and insert conflicts

Figure 8.1 shows the results from an update conflict, which would occur for the value *X* under the following circumstances:

Steps	On Data Store A	On Data Store B
Initial condition	X is 1	X is 1
The application on each data store updates *X* simultaneously	Set X=2	Set X=100
The replication agent on each data store sends its update to the other	Replicate X to data store B	Replicate X to data store A
Each data store now has the other's update	Replication says to set X=100	Replication says to set X=2

Note: Uniqueness conflicts resulting from conflicting inserts follow a similar pattern as update conflicts, only the conflict involves the whole row.

Figure 8.1 Update conflict



If update or insert conflicts remain unchecked, the master and subscriber data stores fall out of synchronization with each other. It may be difficult or even impossible to determine which data store is correct.

With update conflicts, it is possible for a transaction to update many data items but have a conflict on a few of them. Most of the transaction's effects survive the conflict, with only a few being overwritten by replication. If you decide to ignore such conflicts, the transactional consistency of the application data is compromised.

If an update conflict occurs, and if the updated columns for each version of the row are different, then the non-primary key fields for the row may diverge between the replicated tables.

Note: Within a single data store, update conflicts are prevented by the locking protocol: only one transaction at a time can update a specific row in the data store. However, update conflicts can occur in replicated systems due to the ability of each data store to operate independently.

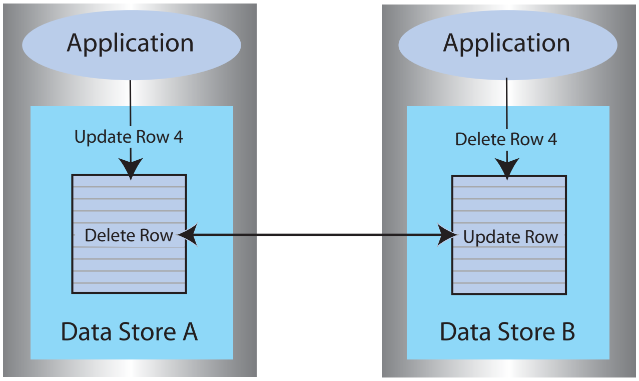
TimesTen replication includes timestamp-based conflict resolution to cope with simultaneous updates or inserts. Through the use of timestamp-based conflict resolution, you may be able to keep the replicated data stores synchronized and transactionally consistent.

Delete/update conflicts

Figure 8.2 shows the results from a delete/update conflict, which would occur for *Row 4* under the following circumstances:

Steps	On Data Store A	On Data Store B
Initial condition	Row 4 exists	Row 4 exists
The applications issue a conflicting update and delete on Row 4 simultaneously	Update Row 4	Delete Row 4
The replication agent on each data store sends the delete or update to the other	Replicate update to data store B	Replicate delete to data store A
Each data store now has the delete or update from the other data store	Replication says to delete Row 4	Replication says to update Row 4

Figure 8.2 Update/Delete conflict



Though TimesTen can detect and report delete/update conflicts, it cannot resolve them. Under these circumstances, the master and subscriber data stores fall out of synchronization with each other.

Though TimesTen cannot ensure synchronization between data stores following such a conflict, it does ensure that the most recent transaction is applied to each data store. If the timestamp for the delete is more recent than that for the update, the row is deleted on each data store. If the timestamp for the update is more recent than that for the delete, the row is updated on the local data store. However, because the row was deleted on the other data store, the replicated update is discarded. See "Reporting delete/update conflicts" on page 179 for example reports.

Note: There is an exception to this behavior when timestamp comparison is enabled on a table using UPDATE BY USER. See "User timestamp column maintenance" on page 173 for details.

Timestamp resolution

For replicated tables that are subject to conflicts, create the table with a special column of type BINARY(8) to hold a timestamp value that indicates the time the row was inserted or last updated. You can then configure TimesTen to automatically insert a timestamp value into this column each time a particular row is changed, as described in "Configuring timestamp comparison" on page 171.

Note: TimesTen does not support conflict resolution between cached tables in a cache group and Oracle.

How replication computes the timestamp column depends on your system:

UNIX

- On UNIX systems, the timestamp value is derived from the **timeval** structure returned by the **gettimeofday** system call. This structure reports the time of day in a pair of 4-byte words to a resolution of 1 microsecond. The actual resolution of the value is system-dependent.

WINDOWS

- On Windows NT systems, the timestamp value is derived from the **GetSystemTimeAsFileTime** Win32 call. The Windows NT file time is reported in units of 0.1 microseconds, but effective granularity can be as coarse as 10 milliseconds.

TimesTen uses the time value returned by the system at the time the transaction performs each update as the record's INSERT or UPDATE time. Therefore, rows that are inserted or updated by a single transaction may receive different timestamp values.

When applying an update received from a master, the replication agent at the subscriber data store performs timestamp resolution in the following manner:

- If the timestamp of the update record is newer than the timestamp of the stored record, TimesTen updates the row. The same rule applies to inserts. If a replicated insert is newer than an existing row, the existing row is overwritten.
- If the timestamp of the update and of the stored record are equal, the update is allowed. The same rule applies to inserts.
- If the timestamp of the update is older than the timestamp of the stored record, the update is discarded. The same rule applies to inserts.
- If a row is deleted, no timestamp is available for comparison. Any update operations on the deleted row are discarded. However, if a row is deleted on one system, then replicated to another system that has more recently updated the row, then the replicated delete is rejected. A replicated insert operation on a deleted row is applied as an insert.
- An update that cannot find the updated row is considered a delete conflict, which is reported but cannot be resolved.

Note: If the ON EXCEPTION NO ACTION option is specified for a table, then the update, insert, or delete that fails a timestamp comparison is rejected. This may result in transactional inconsistencies should replication apply some, but not all, the actions of a transaction. If the ON EXCEPTION ROLLBACK WORK option is specified for a table, an update that fails timestamp comparison causes the entire transaction to be skipped.

Configuring timestamp comparison

To configure timestamp comparison:

- Define a column in your replicated tables to hold the timestamp value.
- Include a CHECK CONFLICTS clause for each TABLE element in your CREATE REPLICATION statement to identify the timestamp column, how timestamps are to be generated, what to do in the event of a conflict, and how to report conflicts.

Establishing a timestamp column in replicated tables

To use timestamp comparison on replicated tables, you must specify a nullable column of type BINARY(8) to hold the timestamp value. The timestamp column must be created along with the table as part of a CREATE TABLE statement—it cannot be added later as part of an ALTER TABLE statement. In addition, the timestamp column cannot be part of a primary key or index. Example 8.1 shows the REP.TAB table contains a column named TSTAMP of type BINARY(8) to hold the timestamp value.

Example 8.1

```
CREATE TABLE REP.TAB (COL1 NUMBER NOT NULL,
                       COL2 NUMBER NOT NULL,
                       TSTAMP BINARY(8),
                       PRIMARY KEY (COL1));
```

If no timestamp column is defined in the replicated table, timestamp comparison cannot be performed to detect conflicts. Instead, at each site, the value of a row in the database reflects the most recent update applied to the row, either by local applications or by replication.

Configuring the CHECK CONFLICTS clause

When configuring your replication scheme, you can set up timestamp comparison for a TABLE element by including a CHECK CONFLICTS clause in the table's ELEMENT description in the CREATE REPLICATION statement.

Note: A CHECK CONFLICT clause cannot be specified for DATASTORE elements.

The syntax of the CREATE REPLICATION statement is described in Chapter 5, "SQL Statements" in the *Oracle TimesTen In-Memory Database SQL Reference Guide*. Below are some examples of how CHECK CONFLICTS might be used when configuring your replication scheme.

Example 8.2

In this example, we establish automatic timestamp comparison for the bidirectional replication scheme shown in Example 3.29 on page 89. The DSNs, WEST_DSN and EAST_DSN, define the WESTDS and EASTDS data stores that replicate the table, REPL.ACCOUNTS, containing the timestamp column, TSTAMP. In the event of a comparison failure, discard the transaction that includes an update with the older timestamp.

```
CREATE REPLICATION REPL.R1
ELEMENT ELEM_ACCOUNTS_1 TABLE REPL.ACCOUNTS
      CHECK CONFLICTS BY ROW TIMESTAMP
          COLUMN TSTAMP
          UPDATE BY SYSTEM
          ON EXCEPTION ROLLBACK WORK
   MASTER WESTDS ON "WESTCOAST"
   SUBSCRIBER EASTDS ON "EASTCOAST"
ELEMENT ELEM_ACCOUNTS_2 TABLE REPL.ACCOUNTS
      CHECK CONFLICTS BY ROW TIMESTAMP
          COLUMN TSTAMP
          UPDATE BY SYSTEM
          ON EXCEPTION ROLLBACK WORK
   MASTER EASTDS ON "EASTCOAST"
   SUBSCRIBER WESTDS ON "WESTCOAST";
```

When bidirectionally replicating data stores with conflict resolution, the replicated tables on each data store must be set with the same CHECK CONFLICTS attributes. If you need to disable or change the CHECK CONFLICTS settings for the replicated tables, use the ALTER REPLICATION statement described in "Eliminating conflict detection" on page 147 and apply to each replicated data store.

System timestamp column maintenance

When timestamp comparison is enabled using:

```
CHECK CONFLICTS BY ROW TIMESTAMP
       COLUMN ColumnName
       UPDATE BY SYSTEM
```

TimesTen automatically maintains the value of the timestamp column using the current time returned by the underlying operating system. This is the default setting.

When you specify UPDATE BY SYSTEM, TimesTen:

- Initializes the timestamp column to the current time when a new record is inserted into the table.
- Updates the timestamp column to the current time when an existing record is modified.

During initial load, the timestamp column values should be left NULL, and applications should not give a value for the timestamp column when inserting or updating a row.

When you use the **ttBulkCp** or **ttMigrate** utility to save TimesTen tables, the saved rows maintain their current timestamp values. When the table is subsequently copied or migrated back into TimesTen, the timestamp column retains the values it had when the copy or migration file was created.

Note: If you configure TimesTen for timestamp comparison *after* using the **ttBulkCp** or **ttMigrate** to copy or migrate your tables, the initial values of the timestamp columns remain NULL, which is considered by replication to be the earliest possible time.

User timestamp column maintenance

When timestamp comparison is enabled on a table using:

```
CHECK CONFLICTS BY ROW TIMESTAMP
       COLUMN ColumnName
       UPDATE BY USER
```

your application is responsible for maintaining timestamp values. The timestamp values used by your application can be arbitrary, but the time values cannot decrease. In cases where the user explicitly sets or updates the timestamp column, the application-provided value is used instead of the current time.

Note: Replicated delete operations always carry a system-generated timestamp. If replication has been configured with UPDATE BY USER and an update/delete conflict occurs, the conflict is resolved by comparing the two timestamp values and the operation with the larger timestamp wins. If the basis for the user timestamp varies from that of the system-generated timestamp, the results may not be as expected. Therefore, if you expect delete conflicts to occur, use system-generated timestamps.

Local updates

To maintain synchronization of tables between replicated sites, TimesTen also performs timestamp comparisons for updates performed by local transactions. If an updated table is declared to have automatic timestamp maintenance, then updates to records that have timestamps exceeding the current system time are prohibited.

Normally, clocks on replicated systems are synchronized sufficiently to ensure that a locally updated record is given a later timestamp than that in the same record stored on the other systems. Perfect synchronization may not be possible or affordable, however. But, by protecting record timestamps from "going backwards," replication can do what is possible to ensure that the tables on replicated systems stay synchronized.

Conflict reporting

TimesTen conflict checking may be configured to report conflicts to a human-readable plain text file, or to an XML file for use by user applications. This section includes the following topics

- Reporting conflicts to a text file
- Reporting conflicts to an XML file
- Reporting uniqueness conflicts
- Reporting update conflicts
- Reporting delete/update conflicts
- Suspending and resuming the reporting of conflicts

Reporting conflicts to a text file

To configure replication to report conflicts to a human-readable text file (the default), use:

```
CHECK CONFLICTS BY ROW TIMESTAMP
   COLUMN ColumnName
   ...
   REPORT TO 'FileName' FORMAT STANDARD
```

An entry is added to the report file *FileName* that describes each conflict. The phrase FORMAT STANDARD is optional and may be omitted, as the standard report format is the default.

Each failed operation logged in the report consists of an entry that starts with a header, followed by information specific to the conflicting operation. Each entry is separated by a number of blank lines in the report.

The header contains:

- The time the conflict was discovered.
- The data stores that sent and received the conflicting update.
- The table in which the conflict occurred.

The header has the following format:

```
Conflict detected at <time> on <date>
Datastore : <subscriber datastore>
Transmitting name : <master datastore>
Table : <username>.<tablename>
```

For example:

```
Conflict detected at 20:08:37 on 05-17-2004
Datastore : /tmp/subscriberds
Transmitting name : MASTERDS
Table : USER1.T1
```

Following the header is the information specific to the conflict. Data values are shown in ASCII format. Binary data is translated into hexadecimal before display, and floating-point values are shown with appropriate precision and scale.

For further description of the conflict report file, see "Reporting uniqueness conflicts" on page 176, "Reporting update conflicts" on page 177 and "Reporting delete/update conflicts" on page 179.

Reporting conflicts to an XML file

To configure replication to report conflicts to an XML file, use:

```
CHECK CONFLICTS BY ROW TIMESTAMP
   COLUMN ColumnName
   ...
   REPORT TO 'FileName' FORMAT XML
```

Replication uses the base file name *FileName* to create two files. `FileName.xml` is a header file that contains the XML Document Type Definition for the conflict report structure, as well as the root element, defined as `<ttrepconflictreport>`. Inside the root element is an XML directive to include the file *FileName*.include, and it is to this file that all conflicts are written. Each conflict is written as a single element of type `<conflict>`.

For further description of the conflict report file XML elements, see "XML Document Type Definition for the Conflict Report File" on page 191.

Note: When performing log maintenance on an XML conflict report file, only the file *FileName*.include should be truncated or moved. For conflict reporting to continue to function correctly, the file *FileName*.xml should be left untouched.

Reporting uniqueness conflicts

A uniqueness conflict record is issued when a replicated INSERT fails because of a conflict.

A uniqueness conflict record in the report file contains:

- The timestamp and values for the *existing tuple*, which is the tuple that the conflicting tuple is in conflict with.
- The timestamp and values for the *conflicting insert tuple*, which is the tuple of the insert that failed.
- The key column values used to identify the record.
- The action that was taken when the conflict was detected (discard the single row insert or the entire transaction)

Note: If the transaction was discarded, the contents of the entire transaction are logged in the report file.

The format of a uniqueness conflict record is:

```
Conflicting insert tuple timestamp : <timestamp in binary format>
Existing tuple timestamp : <timestamp in binary format>
The existing tuple :
<<column value> [,<column value>. ..]>
The conflicting tuple :
<<column value> [,<column value> ...]>
The key columns for the tuple:
<<key column name> : <key column value>>
Transaction containing this insert skipped
Failed transaction:
Insert into table <user>.<table> <<column value> [,<column
value>...]>
End of failed transaction
```

Example 8.3 shows the output from a uniqueness conflict on the row identified by the primary key value, '2'. The older insert replicated from SUBSCRIBERDS conflicts with the newer insert in MASTERDS, so the replicated insert is discarded.

Example 8.3

```
Conflict detected at 13:36:00 on 03-25-2002
Datastore : /tmp/masterds
Transmitting name : SUBSCRIBERDS
Table : REPL.TAB
Conflicting insert tuple timestamp : 3C9F983D00031128
Existing tuple timestamp : 3C9F983E000251C0
The existing tuple :
<     2,       2, 3C9F983E000251C0>
The conflicting tuple :
<     2,     100, 3C9F983D00031128>
The key columns for the tuple:
<COL1 :      2>
Transaction containing this insert skipped
Failed transaction:
Insert into table REPL.TAB <     2,    100, 3C9F983D00031128>
End of failed transaction
```

Reporting update conflicts

An update conflict record is issued when a replicated UPDATE fails because of a conflict. This record reports:

- The timestamp and values for the *existing tuple*, which is the tuple that the conflicting tuple is in conflict with.
- The timestamp and values for the *conflicting update tuple*, which is the tuple of the update that failed.
- The *old values*, which are the original values of the conflicting tuple before the failed update.
- The key column values used to identify the record.
- The action that was taken when the conflict was detected (discard the single row update or the entire transaction).

Note: If the transaction was discarded, the contents of the entire transaction are logged in the report file.

The format of an update conflict record is:

```
Conflicting update tuple timestamp : <timestamp in binary format>
Existing tuple timestamp : <timestamp in binary format>
The existing tuple :
<<column value> [,<column value>. ..]>
The conflicting update tuple :
TSTAMP :<timestamp> :<<column value> [,<column value>. ..]>
The old values in the conflicting update:
TSTAMP :<timestamp> :<<column value> [,<column value>. ..]>
The key columns for the tuple:
<<key column name> : <key column value>>
Transaction containing this update skipped
Failed transaction:
Update table <user>.<table> with keys:
<<key column name> : <key column value>>
New tuple value:
<TSTAMP :<timestamp> :<<column value> [,<column value>. ..]>
End of failed transaction
```

Example 8.4 shows the output from an update conflict on the COL2 value in the row identified by the primary key value, ‘6’. The older update replicated from the MASTERDS data store conflicts with the newer update in SUBSCRIBERDS, so the replicated update is discarded.

Example 8.4

```
Conflict detected at 15:03:18 on 03-25-2002
Datastore : /tmp/subscriberds
Transmitting name : MASTERDS
Table : REPL.TAB
Conflicting update tuple timestamp : 3C9FACB6000612B0
Existing tuple timestamp : 3C9FACB600085CA0
The existing tuple :
<     6,     99, 3C9FACB600085CA0>
The conflicting update tuple :
<TSTAMP :3C9FACB6000612B0, COL2 :    50>
The old values in the conflicting update:
<TSTAMP :3C9FAC85000E01F0, COL2 :     2>
The key columns for the tuple:
<COL1 :     6>
Transaction containing this update skipped
Failed transaction:
Update table REPL.TAB with keys:
<COL1 :     6>
New tuple value: <TSTAMP :3C9FACB6000612B0, COL2 :    50>
End of failed transaction
```

Reporting delete/update conflicts

A delete/update conflict record is issued when an update is attempted on a row that has more recently been deleted. This record reports:

- The timestamp and values for the *conflicting update tuple* or *conflicting delete tuple*, whichever tuple failed.
- If the delete tuple failed, the report also includes the timestamp and values for the *existing tuple*, which is the surviving update tuple with which the delete tuple was in conflict.
- The key column values used to identify the record.
- The action that was taken when the conflict was detected (discard the single row update or the entire transaction).

Note: If the transaction was discarded, the contents of the entire transaction are logged in the report file. TimesTen cannot detect DELETE/INSERT conflicts.

The format of a record that indicates a delete conflict with a failed update is:

```
Conflicting update tuple timestamp : <timestamp in binary format>
The conflicting update tuple :
TSTAMP :<timestamp> :<<column value> [,<column value>. ..]>
This transaction skipped
The tuple does not exist
Transaction containing this update skipped
Update table <user>.<table> with keys:
<<key column name> : <key column value>>
New tuple value:
<TSTAMP :<timestamp> :<<column value> [,<column value>. ..]>
End of failed transaction
```

Example 8.5 shows the output from a delete/update conflict caused by an update on a row that has more recently been deleted. Because there is no row to update, the update from SUBSCRIBERDS is discarded.

Example 8.5

```
Conflict detected at 15:27:05 on 03-25-2002
Datastore : /tmp/masterds
Transmitting name : SUBSCRIBERDS
Table : REPL.TAB
Conflicting update tuple timestamp : 3C9FB2460000AFC8
The conflicting update tuple :
<TSTAMP :3C9FB2460000AFC8, COL2 :    99>
The tuple does not exist
Transaction containing this update skipped
Failed transaction:
Update table REPL.TAB with keys:
<COL1 :     2>
New tuple value: <TSTAMP :3C9FB2460000AFC8,
 COL2 :    99>
End of failed transaction
```

The format of a record that indicates an update conflict with a failed delete is:

```
Conflicting binary delete tuple timestamp : <timestamp in binary
format>Example 8.7
Existing binary tuple timestamp : <timestamp in binary format>
The existing tuple :
<<column value> [,<column value>. ..]>
The key columns for the tuple:
<<key column name> : <key column value>>
Transaction containing this delete skipped
Failed transaction:
Delete table <user>.<table> with keys:
<<key column name> : <key column value>>
End of failed transaction
```

Example 8.6 shows the output from a delete/update conflict caused by a delete on a row that has more recently been updated. Because the row was updated more recently than the delete, the delete from MASTERDS is discarded.

Example 8.6

```
Conflict detected at 15:27:20 on 03-25-2002
Datastore : /tmp/subscriberds
Transmitting name : MASTERDS
Table : REPL.TAB
Conflicting binary delete tuple timestamp : 3C9FB258000708C8
Existing binary tuple timestamp : 3C9FB25800086858
The existing tuple :
<   147,     99, 3C9FB25800086858>
The key columns for the tuple:
<COL1 :   147>
Transaction containing this delete skipped
Failed transaction:
Delete table REPL.TAB with keys:
<COL1 :   147>
```

Suspending and resuming the reporting of conflicts

Provided your applications are well-behaved, replication usually encounters and reports only sporadic conflicts. However, it is sometimes possible under heavy load to trigger a flurry of conflicts in a short amount of time, particularly when applications are in development and such errors are expected. This can potentially have a negative impact on the performance of the host machine because of excessive writes to the conflict report file and the large number of SNMP traps that can be generated.

To avoid overwhelming a host with replication conflicts, you may configure replication to suspend conflict reporting when the number of conflicts per second has exceeded a user-specified threshold. Conflict reporting may also be configured to resume once the conflicts per second have fallen below a user-specified threshold.

Conflict reporting suspension and resumption can be detected by an application by catching the SNMP traps **ttRepConflictReportStoppingTrap** and **ttRepConflictReportStartingTrap**, respectively. See Chapter 3, "Diagnostics through SNMP Traps" in *Oracle TimesTen In-Memory Database Error Messages and SNMP Traps* for more information.

To configure conflict reporting to be suspended and resumed based on the number of conflicts per second, use the CONFLICT REPORTING SUSPEND AT and CONFLICT REPORTING RESUME AT attributes for the STORE clause of a replication scheme. Example 8.7 demonstrates the configuration of a replication schemes where conflict reporting ceases when the number of conflicts exceeds 20 per second, and conflict reporting resumes when the number of conflicts drops below 10 per second.

Note: When conflict reporting is suspended, the SNMP trap **ttRepConflictReportStartingTrap** is not sent immediately when the number of conflicts per second drops to zero. If the number of conflicts per second drops quickly to zero after a burst of conflicts, **ttRepConflictReportStartingTrap** is only sent when the next conflict is reported.

Note: If the replication agent is stopped while conflict reporting is suspended, conflict reporting is enabled when the replication agent is restarted. The SNMP trap **ttRepConflictReportingStartingTrap** is not sent if this occurs. This means that an application that monitors the conflict report suspension traps must also monitor the traps for replication agent stopping and starting.

Example 8.7

```
CREATE REPLICATION REPL.R1
ELEMENT ELEM_ACCOUNTS_1 TABLE REPL.ACCOUNTS
      CHECK CONFLICTS BY ROW TIMESTAMP
          COLUMN TSTAMP
          UPDATE BY SYSTEM
          ON EXCEPTION ROLLBACK WORK
          REPORT TO 'conflicts' FORMAT XML
   MASTER WESTDS ON "WESTCOAST"
   SUBSCRIBER EASTDS ON "EASTCOAST"
ELEMENT ELEM_ACCOUNTS_2 TABLE REPL.ACCOUNTS
      CHECK CONFLICTS BY ROW TIMESTAMP
          COLUMN TSTAMP
          UPDATE BY SYSTEM
          ON EXCEPTION ROLLBACK WORK
          REPORT TO 'conflicts' FORMAT XML
   MASTER EASTDS ON "EASTCOAST"
   SUBSCRIBER WESTDS ON "WESTCOAST"
STORE WESTDS ON "WESTCOAST"
   CONFLICT REPORTING SUSPEND AT 20
   CONFLICT REPORTING RESUME AT 10
STORE EASTDS ON "EASTCOAST”
   CONFLICT REPORTING SUSPEND AT 20
   CONFLICT REPORTING RESUME AT 10;
```

Managing data store failover and recovery

As discussed in "Designing a highly available system" on page 39, a fundamental element in the design of a highly available system is the ability to quickly recover from a failure. Failures may be related to:

Hardware Problems:

- System failure
- Network failure

Software Problems:

- Operating system failure
- Application failure
- Data store failure
- Operator error

Your replicated system must employ a "cluster manager" or custom software to detect such failures and, in the event of a failure involving a master data store, redirect the user load to one of its subscribers. TimesTen does not provide a cluster manager or make any assumptions about how they operate, so the focus of this discussion is on the TimesTen mechanisms that an application or cluster manager can use to recover from failures.

Unless the replication scheme is configured to use the return twosafe service, TimesTen replicates updates only after the original transaction commits to the master data store. If a subscriber data store is inoperable or communication to a subscriber data store fails, updates at the master are not impeded. During outages at subscriber systems, updates intended for the subscriber are saved in the TimesTen transaction log.

Note: If TimesTen was installed with Access Control enabled, most of the procedures described in this section require that you have ADMIN privileges to the data store. See Chapter 1, "Access Control" in the *Oracle TimesTen In-Memory Database Installation Guide* for details.

General failover and recovery procedures

The procedures for managing failover and recovery depend primarily on:

- Your replication scheme, as described in "Failover and recovery" on page 40.
- Whether the failure occurred on a master or subscriber data store.
- Whether the threshold for the transaction log on the master is exhausted before the problem is resolved and the data stores reconnected.

Subscriber failures

If your replication scheme is configured for default asynchronous replication, should a subscriber data store become inoperable or communication to a subscriber data store fail, updates at the master are not impeded and the cluster manager does not have to take any immediate action.

Note: If the failed subscriber is configured to use a return service, you must first disable return service blocking, as described in "Managing return service timeout errors and replication state changes" on page 66.

During outages at subscriber systems, updates intended for the subscriber are saved in the transaction log on the master. If the subscriber agent reestablishes communication with its master before the master reaches its FAILTHRESHOLD, the updates held in the log are automatically transferred to the subscriber and no further action is required. (See "Setting the log failure threshold" on page 108 for details on how to establish the FAILTHRESHOLD value for the master data store.)

If the FAILTHRESHOLD is exceeded, the master sets the subscriber to the **Failed** state and it must be recovered, as described in "Recovering a failed data store" on page 187. Any application that connects to the failed subscriber receives a *tt_ErrReplicationInvalid* (8025) warning indicating that the data store has been marked **Failed** by a replication peer.

Applications can use the ODBC **SQLGetInfo** function to check if the subscriber data store it is connected to has been set to the **Failed** state. The **SQLGetInfo** function includes a TimesTen-specific infotype, TT_REPLICATION_INVALID, that returns a 32-bit integer value of '1' if the data store is failed, or '0' if not failed. Since the infotype TT_REPLICATION_INVALID is specific to TimesTen, all applications using it need to include the `timesten.h` file in addition to the other ODBC include files.

Example 8.8 For example, to check if the data store identified by the *hdbc* handle has been set to the Failed state:

```
SQLINTEGER retStatus;

SQLGetInfo(hdbc, TT_REPLICATION_INVALID,
          (PTR)&retStatus, NULL, NULL);
```

Master failures

The cluster manager plays a more central role if a failure involves the master data store. Should a master data store fail, the cluster manager must detect this event and redirect the user load to one of its surviving data stores. This surviving subscriber then becomes the master, which continues to accept transactions and replicates them to the other surviving subscriber data stores. If the failed master and surviving subscriber are configured in a bidirectional manner, transferring the user load from a failed master to a subscriber does not require that you make any changes to your replication scheme. However, when using unidirectional replication or complex schemes, such as those involving propagators, you may have to issue one or more ALTER REPLICATION statements to reconfigure the surviving subscriber as the “new master” in your scheme. See “Replacing a master data store” on page 147 for an example.

When the problem is resolved, if you are not using the hot-standby configuration or the active standby pair described in “Automatic catch-up of a failed master data store” on page 185, you must recover the master data store as described in “Recovering a failed data store” on page 187.

After the data store is back online, the cluster manager can either transfer the user load back to the original master or reestablish it as a subscriber for the “acting master.” See “Failover and recovery” on page 40 for more information.

Automatic catch-up of a failed master data store

The master catch-up feature automatically restores a failed master data store from a subscriber data store without the need to invoke the **ttRepAdmin** `-duplicate` operation described in “Recovering a failed data store” on page 187.

The master catch-up feature needs no configuration, but it can be used only in the following types of configurations:

- A single master replicated in a bidirectional, hot-standby manner to a single subscriber
- An active standby pair in which the active master data store is replicated to the standby data store which then propagates changes to up to 127 read-only subscribers

In addition, the following must be true:

- The ELEMENT type is DATASTORE.

- TRANSMIT NONDURABLE or RETURN TWOSAFE must enabled. TRANSMIT NONDURABLE is optional for asynchronous and return receipt transactions.

When the master replication agent is restarted after a crash or invalidation, any lost transactions that originated on the master are automatically reapplied from the subscriber to the master. No connections are allowed to the master store until it has completely caught up with the subscriber. Applications attempting to connect to a data store during the catch-up phase receive an error that indicates a catch-up is in progress. The only exception is if you connect to a data store with the **ForceConnect** attribute set in the DSN.

When the catch-up phase is complete, your application can connect to the data store. An SNMP trap and message to the system log indicate the completion of the catch-up phase.

If one of the stores is invalidated or crashes during the catch-up process, the catch-up phase is resumed when the store comes back up.

Master/subscriber failures

As described in "Unidirectional or bidirectional replication" on page 19, you can distribute the workload over multiple bidirectionally replicated data stores, each of which serves as both master and subscriber. When recovering a master/subscriber data store, the log on the failed data store may present problems when you restart replication.

If a data store in a distributed workload scheme fails and work is shifted to a surviving data store, the information in the surviving data store becomes more current than that in the failed data store. If replication is restarted at the failed system before the FAILTHRESHOLD has been reached on the surviving data store, then both data stores attempt to update one another with the contents of their transaction logs. In this case, the older updates in the transaction log on the failed data store may overwrite more recent data on the surviving system.

There are two ways to recover in such a situation:

- If the timestamp conflict resolution rules described in "Replication conflict detection and resolution" on page 167 are sufficient to guarantee consistency for your application, then you can restart the failed system and allow the updates from the failed data store to propagate to the surviving data store. The conflict resolution rules prevent more recent updates from being overwritten.
- Recreate the failed data store, as described in "Recovering a failed data store" on page 187.

Note: If the data store must be recreated, the updates in the log on the failed data store that were not received by the surviving data store cannot be identified or restored. In the case of several surviving data stores, you must select which of the surviving data stores is to be used to recreate the failed data store. It is possible that at the time the failed data store is recreated, that the selected surviving data store may not have received all updates from the other surviving data stores. This results in diverging data stores. The only way to prevent this situation is to recreate the other surviving data stores from the selected surviving data store.

Network failures

In the event of a temporary network failure, you need not perform any specific action to continue replication. The replication agents that were in communication attempt to reconnect every few seconds. Should the agents reconnect before the master data store runs out of log space, the replication protocol makes sure they neither miss nor repeat any replication updates. If the network is unavailable for a longer period and the FAILTHRESHOLD has been exceeded for the master log, you need to recover the subscriber as described in "Recovering a failed data store" on page 187.

Failures involving sequences

After a link failure, if replication is allowed to recover by replaying queued logs, you do not need to take any action.

However, if the failed node was down for a significant amount of time, you must use the **ttRepAdmin** `-duplicate` command to repopulate the data store on the failed node with transactions from the surviving node, as sequences are not rolled back during failure recovery. In this case, the **ttRepAdmin** `-duplicate` command copies the sequence definitions from one node to the other.

Recovering a failed data store

If a restarted data store cannot be recovered from its master's transaction log so that it is consistent with the other data stores in the replicated system, you must recreate the data store from one of its replication peers. If your data stores are configured in a hot-standby replication scheme, as described in "Automatic catch-up of a failed master data store" on page 185, a failed master data store is automatically brought up to date from the subscriber. Data stores configured with other types of replication schemes must be restored using command line utilities or programmatically using the TimesTen Utility C functions, as described below.

Note: It is not necessary to recreate the DSN for the failed data store.

In the event of a subscriber failure, if any tables are configured with a return service, commits on those tables in the master data store are blocked until the return service time-out period expires. To avoid this, you can establish a return service failure and recovery policy in your replication scheme, as described in "Managing return service timeout errors and replication state changes" on page 66. If you are using the RETURN RECEIPT service, an alternative is to use ALTER REPLICATION and set the NO RETURN attribute to disable return receipt until the subscriber is restored and caught up. Then, you can submit another ALTER REPLICATION to re-establish RETURN RECEIPT.

From the command line

If the data stores are fully replicated, you can use **ttDestroy** to remove the failed data store from memory and **ttRepAdmin** `-duplicate` to recreate it from a surviving data store. If the data store contains any cache groups, you must also use the `-keepCG` option of **ttRepAdmin**.

Example 8.9

For example, to recover a failed data store, *subscriberds*, from a master, named *masterds* on host *system1*, enter:

```
> ttDestroy /tmp/subscriberds
```

```
> ttRepAdmin -dsn subscriberds -duplicate -from masterds
-host "system1"
```

Note: **ttRepAdmin** `-duplicate` is only supported between identical and patch TimesTen releases (the major and minor release numbers must be the same).

After recreating the data store with **ttRepAdmin** `-duplicate`, the first connection to the data store reloads it into memory. To improve performance when duplicating large data stores, you can avoid the reload step by using the **ttRepAdmin** `-ramLoad` option to keep the data store in memory after the duplicate operation.

Example 8.10

For example, to recover a failed data store, *subscriberds*, from a master, named *masterds* on host *system1*, and to keep the data store in memory and restart replication after the duplicate operation, enter:

```
> ttDestroy /tmp/subscriberds
> ttRepAdmin -dsn subscriberds -duplicate -ramLoad -from masterds
-host "system1" -setMasterRepStart
```

Note: After duplicating a data store with the **ttRepAdmin** `-duplicate` `-ramLoad` options, the RAM Policy for the data store is **manual** until explicitly reset by **ttAdmin** `-ramPolicy` or the **ttRamPolicy** function.

From a program

You can use the C functions provided in the TimesTen Utility library to programmatically recover a failed data store.

If the data stores are fully replicated, you can use **ttDestroyDataStore** function to remove the failed data store and the **ttRepDuplicateEx** function to recreate it from a surviving data store.

Example 8.11 For example, to recover and start a failed data store, named *subscriberds* on host *system2*, from a master, named *masterds* on host *system1*, enter:

```
int            rc;
ttUtilHandle  utilHandle;
ttRepDuplicateExArg arg;
memset( &arg, 0, sizeof( arg ) );
arg.size = sizeof( ttRepDuplicateExArg );
arg.flags = TT_REPDUP_REPSTART | TT_REPDUP_RAMLOAD;
arg.localHost = "system2";
rc = ttDestroyDataStore( utilHandle, "subscriberds", 30 );
rc = ttRepDuplicateEx( utilHandle, "DSN=subscriberds",
                       "masterds", "system1", &arg );
```

In this example, the timeout for the **ttDestroyDataStore** operation is 30 seconds. The last parameter of the **ttRepDuplicateEx** function is an argument structure containing two flags--TT_REPDUP_RESTART to set the *subscriberds* data store to the **Start** state after the duplicate operation is completed, and TT_REPDUP_RAMLOAD to set the RAM Policy to **manual** and keep the data store in memory.

Note: When the TT_REPDUP_RAMLOAD flags is used with **ttRepDuplicateEx,** the RAM policy for the duplicate data store is **manual** until explicitly reset by the **ttRamPolicy** function or **ttAdmin** `-ramPolicy`.

See Chapter 6, "TimesTen Utility API" in the *Oracle TimesTen In-Memory Database C Developer's and Reference Guide* for the complete list of the functions provided in the TimesTen C Language Utility Library.

Recovering NONDURABLE data stores

If your data store is configured with the TRANSMIT NONDURABLE option in a hot-standby configuration, as described in "Automatic catch-up of a failed master data store" on page 185, you do not need to take any action to recover a failed master data store.

For other types of configurations, if the master data store configured with the TRANSMIT NONDURABLE option fails, you must use **ttRepAdmin** `-duplicate` or **ttRepDuplicateEx** to recreate the master data store from the most current subscriber data store. If your application attempts to reconnect to the master store without first performing the duplicate operation, the replication agent recovers the data store, but any attempt to connect results in an error that advises you to perform the 'duplicate'. To avoid this error, your application must reconnect with the connection attribute, **ForceConnect** set to 1.

Writing a failure recovery script

Upon detecting a failure, the cluster manager should invoke a script that effectively executes the procedure shown by the pseudocode in Example 8.12.

Example 8.12

```
Detect problem {
       if (Master == unavailable) {
          FailedDataStore = Master
          FailedDSN = Master_DSN
          SurvivorDataStore = Subscriber
          switch users to SurvivorDataStore
       }
       else {
          FailedDataStore = Subscriber
          FailedDSN = Subscriber_DSN
          SurvivorDataStore = Master
       }
}
Fix problem....
If (Problem resolved) {
       Get state for FailedDataStore
       if (state == "failed") {
          ttDestroy FailedDataStore
          ttRepAdmin -dsn FailedDSN -duplicate
                    -from SurvivorDataStore -host SurvivorHost
                    -setMasterRepStart
       }
       else {
          ttAdmin -repStart FailedDSN
       }
       while (backlog != 0) {
          wait
       }
}
Switch users back to Master
```

This applies to either the master or subscriber data stores. If the master fails, you may lose some transactions.

9

XML Document Type Definition for the Conflict Report File

This chapter describes the Document Type Definition (DTD) and structure of an XML format replication conflict report file. The TimesTen XML format conflict report is are based on the XML 1.0 specification (http://www.w3.org/TR/REC-xml). For information on configuring replication to report conflicts, see "Replication conflict detection and resolution" on page 167.

This chapter includes:

- The conflict report XML Document Type Definition
- The main body of the document
- The uniqueness conflict element
- The update conflict element
- The delete/update conflict element

The conflict report XML Document Type Definition

The XML Document Type Definition (DTD) for the replication conflict report is a set of markup declarations that describes the elements and structure of a valid XML file containing a log of replication conflicts. This DTD can be found in the XML header file—the file with the suffix ".xml"—that is created when replication is configured to report conflicts to an XML file. User applications which understand XML use the DTD to parse the rest of the XML replication conflict report. For more information on reading and understanding XML Document Type Definitions, see http://www.w3.org/TR/REC-xml.

```
<?xml version="1.0"?>
<!DOCTYPE ttreperrorlog [
    <!ELEMENT ttrepconflictreport(conflict*) >
    <!ELEMENT repconflict          header, conflict) >
    <!ELEMENT header              (time, datastore, transmitter, table) >
    <!ELEMENT time                (hour, min, sec, year, month, day) >
    <!ELEMENT hour                (#PCDATA) >
    <!ELEMENT min                 (#PCDATA) >
    <!ELEMENT sec                 (#PCDATA) >
    <!ELEMENT year                (#PCDATA) >
    <!ELEMENT month               (#PCDATA) >
    <!ELEMENT day                 (#PCDATA) >
    <!ELEMENT datastore           (#PCDATA) >
    <!ELEMENT transmitter         (#PCDATA) >
    <!ELEMENT table               (tableowner, tablename) >
    <!ELEMENT tableowner          (#PCDATA) >
    <!ELEMENT tablename           (#PCDATA) >
    <!ELEMENT scope               (#PCDATA) >
    <!ELEMENT failedtransaction   ((insert | update | delete)+) >
    <!ELEMENT insert              (sql) >
    <!ELEMENT update              (sql, keyinfo, newtuple) >
    <!ELEMENT delete              (sql, keyinfo) >
    <!ELEMENT sql                 (#PCDATA) >
    <!ELEMENT keyinfo             (column+) >
    <!ELEMENT newtuple            (column+) >
    <!ELEMENT column              (columnname, columntype, columnvalue) >
    <!ATTLIST column
        pos CDATA #REQUIRED >
    <!ELEMENT columnname          (#PCDATA) >
    <!ELEMENT columnvalue         (#PCDATA) >
    <!ATTLIST columnvalue
        isnull (true | false) "false"
    >
    <!ELEMENT existingtuple       (column+) >
    <!ELEMENT conflictingtuple    (column+) >
    <!ELEMENT conflictingtimestamp(#PCDATA) >
    <!ELEMENT existingtimestamp    (#PCDATA) >
    <!ELEMENT oldtuple            (column+) >
    <!ELEMENT conflict            (conflictingtimestamp, existingtimestamp*,
                                       existingtuple*, existingtimestamp*,
                                       conflictingtuple*, oldtuple*, keyinfo*) >
    <!ATTLIST conflict
       type (insert | update | deletedupdate | updatedeleted) #REQUIRED
    >
    <!ENTITY logFile              SYSTEM "Filename.include">
]>
<ttrepconflictreport>
    &logFile;
</ttrepconflictreport>
```

The main body of the document

The .xml file for the XML replication conflict report is merely a header, containing the XML Document Type Definition describing the report format as well as a link to a file with the suffix ".include". This include file is the main body of the report, containing each replication conflict as a separate element. There are three possible types of elements: insert, update and delete/update conflicts. Each conflict type requires a slightly different element structure.

The uniqueness conflict element

A uniqueness conflict occurs when a replicated insertion fails because a row with an identical key column was inserted more recently. See "Reporting uniqueness conflicts" on page 176 for a description of the information that is written to the conflict report for a uniqueness conflict.

Example 9.1 illustrates the format of a uniqueness conflict XML element, using the values from Example 8.3:

Example 9.1

```
<repconflict>
    <header>
      <time>
          <hour>13</hour>
          <min>36</min>
          <sec>00</sec>
          <year>2002</year>
          <month>03</month>
          <day>25</day>
      </time>
      <datastore>/tmp/masterds</datastore>
      <transmitter>SUBSCRIBERDS</transmitter>
      <table>
          <tableowner>REPL</tableowner>
          <tablename>TAB</tablename>
      </table>
    </header>
    <conflict type="insert">
      <conflictingtimestamp>3C9F983D00031128</conflictingtimestamp>
      <existingtimestamp>3C9F983E000251C0</existingtimestamp>
      <existingtuple>
          <column pos="1">
            <columnname>COL1</columnname>
            <columntype>NUMBER(38)</columntype>
            <columnvalue>2</columnvalue>
          </column>
          <column pos="2">
             <columnname>COL2</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>2</columnvalue>
          </column>
             <columnname>TSTAMP</columnname>
             <columntype>BINARY(8)</columntype>
             <columnvalue>3C9F983E000251C0</columnvalue>
          </column>
       </existingtuple>
       <conflictingtuple>
          <column pos="1">
             <columnname>COL1</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>2</columnvalue>
          </column>
          <column pos="2">
             <columnname>COL2</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>100</columnvalue>
          </column>
          <column pos="3">
            <columnname>TSTAMP</columnname>
            <columntype>BINARY(8)</columntype>
            <columnvalue>3C9F983D00031128</columnvalue>
          </column>
       </conflictingtuple>
       <keyinfo>
          <column pos="1">
            <columnname>COL1</columnname>
            <columntype>NUMBER(38)</columntype>
            <columnvalue>2</columnvalue>
          </column>
       </keyinfo>
    </conflict>
    <scope>TRANSACTION</scope>
    <failedtransaction>
       <insert>
          <sql>Insert into table REPL.TAB </sql>
          <column pos="1">
             <columnname>COL1</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>2</columnvalue>
          </column>
```

```
          <column pos="2">
             <columnname>COL2</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>100</columnvalue>
          </column>
          <column pos="3">
             <columnname>TSTAMP</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>3C9F983D00031128</columnvalue>
          </column>
       </insert>
    </failedtransaction>
</repconflict>
```

The update conflict element

An update conflict occurs when a replicated update fails because the row was updated more recently. See “Reporting update conflicts” on page 177 for a description of the information that is written to the conflict report for an update conflict.

Example 9.2 illustrates the format of an update conflict XML element, using the values from Example 8.4:

Example 9.2

```
<repconflict>
    <header>
       <time>
          <hour>15</hour>
          <min>03</min>
          <sec>18</sec>
          <year>2002</year>
          <month>03</month>
          <day>25</day>
       </time>
       <datastore>/tmp/subscriberds</datastore>
       <transmitter>MASTERDS</transmitter>
       <table>
          <tableowner>REPL</tableowner>
          <tablename>TAB</tablename>
       </table>
    </header>
    <conflict type="update">
       <conflictingtimestamp>
          3C9FACB6000612B0
       </conflictingtimestamp>
       <existingtimestamp>3C9FACB600085CA0</existingtimestamp>
```

```
   <existingtuple>
      <column pos="1">
         <columnname>COL1</columnname>
         <columntype>NUMBER(38)</columntype>
         <columnvalue>6</columnvalue>
      </column>
      <column pos="2">
         <columnname>COL2</columnname>
         <columntype>NUMBER(38)</columntype>
         <columnvalue>99</columnvalue>
      </column>
      <column pos="3">
         <columnname>TSTAMP</columnname>
         <columntype>BINARY(8)</columntype>
         <columnvalue>3C9FACB600085CA0></columnvalue>
      </column>
   </existingtuple>
   <conflictingtuple>
      <column pos="3">
         <columnname>TSTAMP</columnname>
         <columntype>BINARY(8)</columntype>
         <columnvalue>3C9FACB6000612B0</columnvalue>
      </column>
      <column pos="2">
         <columnname>COL2</columnname>
         <columntype>NUMBER(38)</columntype>
         <columnvalue>50</columnvalue>
      </column>
   </conflictingtuple>
   <oldtuple>
      <column pos="3">
         <columnname>TSTAMP</columnname>
         <columntype>BINARY(8)</columntype>
         <columnvalue>3C9FAC85000E01F0</columnvalue>
      </column>
      <column pos="2">
         <columnname>COL2</columnname>
         <columntype>NUMBER(38)</columntype>
         <columnvalue>2</columnvalue>
      </column>
   </oldtuple>
   <keyinfo>
      <column pos="1">
         <columnname>COL1</columnname>
         <columntype>NUMBER(38)</columntype>
         <columnvalue>6</columnvalue>
      </column>
   </keyinfo>
</conflict>
```

```
    <scope>TRANSACTION</scope>
    <failedtransaction>
       <update>
          <<sql>Update table REPL.TAB</sql>
          <<keyinfo>
             <column pos="1">
                <columnname>COL1</columnname>
                <columntype>NUMBER(38)</columntype>
                <columnvalue>6</columnvalue>
             </column>
          </keyinfo>
          <column pos="3">
             <columnname>TSTAMP</columnname>
             <columntype>BINARY(8)</columntype>
             <columnvalue>3C9FACB6000612B0</columnvalue>
          </column>
          <column pos="2">
             <columnname>COL2</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>50</columnvalue>
          </column>
       </update>
    </failedtransaction>
</repconflict>
```

The delete/update conflict element

A delete/update conflict occurs when a replicated update fails because the row to be updated has already been deleted on the data store receiving the update, or when a replicated deletion fails because the row has been updated more recently. See “Reporting delete/update conflicts” on page 179 for a description of the information that is written to the conflict report for a delete/update conflict.

Example 9.3 illustrates the format of a delete/update conflict XML element in which an update fails because the row has been deleted more recently, using the values from Example 8.5:

Example 9.3

```
<repconflict>
   <header>
       <time>
          <hour>15</hour>
          <min>27</min>
          <sec>05</sec>
          <year>2002</year>
          <month>03</month>
          <day>25</day>
       </time>
       <datastore>/tmp/masterds</datastore>
```

```
    <transmitter>SUBSCRIBERDS</transmitter>
    <table>
       <tableowner>REPL</tableowner>
       <tablename>TAB</tablename>
    </table>
</header>
<conflict type="update">
    <conflictingtimestamp>
       3C9FB2460000AFC8
    </conflictingtimestamp>
    <conflictingtuple>
       <column pos="3">
          <columnname>TSTAMP</columnname>
          <columntype>BINARY(8)</columntype>
          <columnvalue>3C9FB2460000AFC8</columnvalue>
       </column>
       <column pos="2">
          <columnname>COL2</columnname>
          <columntype>NUMBER(38)</columntype>
          <columnvalue>99/columnvalue>
       </column>
    </conflictingtuple>
    <keyinfo>
       <column pos="1">
          <columnname>COL1</columnname>
          <columntype>NUMBER(38)</columntype>
          <columnvalue>2</columnvalue>
       </column>
    </keyinfo>
 </conflict>
 <scope>TRANSACTION</scope>
 <failedtransaction>
    <update>
       <sql>Update table REPL.TAB</sql>
```

```
          <keyinfo>
             <column pos="1">
                <columnname>COL1</columnname>
                <columntype>NUMBER(38)</columntype>
                <columnvalue>2</columnvalue>
             </column>
          </keyinfo>
          <column pos="3">
             <columnname>TSTAMP</columnname>
             <columntype>BINARY(8)</columntype>
             <columnvalue>3C9FB2460000AFC8</columnvalue>
          </column>
          <column pos="2">
             <columnname>COL2</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>99</columnvalue>
          </column>
         </update>
   </failedtransaction>
</repconflict>
```

Example 9.4 illustrates the format of a delete/update conflict XML element in which a deletion fails because the row has been updated more recently, using the values from Example 8.6:

Example 9.4

```
<repconflict>
   <header>
       <time>
          <hour>15</hour>
          <min>27</min>
          <sec>20</sec>
          <year>2002</year>
          <month>03</month>
          <day>25</day>
       </time>
       <datastore>/tmp/masterds</datastore>
       <transmitter>MASTERDS</transmitter>
       <table>
          <tableowner>REPL</tableowner>
          <tablename>TAB</tablename>
       </table>
   </header>
   <conflict type="delete">
       <conflictingtimestamp>
          3C9FB258000708C8
       </conflictingtimestamp>
       <existingtimestamp>3C9FB25800086858</existingtimestamp>
       <existingtuple>
          <column pos="1">
             <columnname>COL1</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>147</columnvalue>
          </column>
          <column pos="2">
             <columnname>COL2</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>99</columnvalue>
          </column>
          <column pos="3">
             <columnname>TSTAMP</columnname>
             <columntype>BINARY(8)</columntype>
             <columnvalue>3C9FB25800086858</columnvalue>
          </column>
       </existingtuple>
       <keyinfo>
          <column pos="1">
             <columnname>COL1</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>147</columnvalue>
          </column>
       </keyinfo>
    </conflict>
    <scope>TRANSACTION</scope>
    <failedtransaction>
       <delete>
          <sql>Delete from table REPL.TAB</sql>
          <keyinfo>
             <column pos="1">
                <columnname>COL1</columnname>
                <columntype>NUMBER(38)</columntype>
                <columnvalue>147</columnvalue>
             </column>
          </keyinfo>
       </delete>
    </failedtransaction>
</repconflict>
```

Glossary

asynchronous replication

A replication technique where one replica (or a proper subset of replicas) is updated in an initial transaction. The update is asynchronously propagated to other replicas after the initial updating transaction is 1-safe committed. Finally, the updates are applied to the replicas in refresh transactions. The propagating transactions and refresh transactions are typically separate from each other, and separate for each replica. Some protocols coordinate the propagation and refresh activity.

backup copy

A secondary copy of a replication element. Usually used in connection with a single secondary copy in a double-contingency scheme, or two secondary copies in a triple-contingency scheme.

backup data store (node)

A subscriber data store (node) in a replication scheme.

bidirectional replication

A replication configuration in which two different data stores transmit updates to each other.

latency

In the context of asynchronous replication, the mean over time of (the time difference between the commit of an application's update transaction on primary copies and the commit of the corresponding refresh transaction(s) on secondary copies). Latency is a measure of replication performance.

master copy

See primary copy.

master data store (node)

A data store (node) that owns a replication element. That is, it is granted the capability of updating its (primary) copy of the replication element. Contrast “subscriber data store (node)”. A data store may be both a master and a subscriber - for different replication elements.

master/subscriber data store (node)

A data store that is both the master (holding a primary copy) for some replication elements and the subscriber (holding a secondary copy) for others.

master replication

A replication rule whereby for each replication element, the members of some subset of data stores (typically a singleton) are designated owners. The replica in each of these stores is distinguished as a “primary” copy that always contains the correct current value. An application can directly update only this copy. All other replicas are secondary copies and (at best) read-only. TimesTen supports this model, but also permits bidirectional replication, which permits updates to either data store.

missing transaction problem

The loss of transaction updates, hence, the loss of the containing transaction as the result of the failure of a master replica in a 1-safe replication scheme. Every 1-safe scheme may possibly lose committed but incompletely transmitted transactions, as well as any dependent transactions (minimal divergence). Some schemes (for example, ones that run an epoch algorithm) may lose more than this.

primary copy

An updatable copy of a replication element. Also called master copy.

refresh transaction

A transaction that a replication facility runs to install updates on subscribers. The correspondence between application transactions and refresh transactions may be one-to-many, one-to-one, or many-to-one. In an asynchronous replication scheme refresh transactions are separate from the original (application) update transaction on primary replicas.

replica

Either a primary or secondary copy of a replication element.

replication agent

Replication at each master and subscriber data store is controlled by a *replication agent.* The replication agent on the master data store reads the records from the transaction log and forwards any detected changes to replicated elements to the replication agent on the subscriber data store. The replication agent on the subscriber then applies the updates to its data store.

replication configuration

The assignment of primary and secondary replicas to data stores in a replication group. In a lazy master replication scheme this assignment implicitly defines a directed graph whose nodes are data stores and with a directed edge from each node representing a master store to each node representing a slave store that contains a secondary copy of a replication element whose primary copy is in the master.

replication (data) set

A set of replication elements or data partitions that participate in a replication scheme.

replication element

An entity that TimesTen synchronizes between data stores. At this time, TimesTen supports data stores, sequences and complete tables as replication elements.

replication scheme

The definition of a set of replication elements or partitions that comprise a replication data set, a set of data stores (replication group) that house the various replica copies of elements in the replication set, the assignment of primary and secondary copies of each replication element to data stores in the replication group which implicitly defines the replication configuration or topology, and whether the replication is “full” or “selective”, a propagation rule that defines how and when updates are transmitted, a refresh rule that defines how and when secondary copies are updated, and a set of rules that govern the usage of secondary copies.

A replication scheme may include additional attributes, for example a recovery discipline for single or multiple failures.

secondary copy

A non-updatable copy of a replication element. Also called backup copy.

selective replication

A replication scheme in which different stores have different sets of replication elements. In such a scheme, the master store for each replication element selectively transmits its updates to a proper subset of the replication group's slaves. Selective replication complicates recovery.

Index