Oracle® TimesTen In-Memory Database

Replication Guide

11*g* Release 2 (11.2.2)

E21635-04

September 2012

ORACLE®

Oracle TimesTen In-Memory Database Replication Guide, 11*g* Release 2 (11.2.2)

E21635-04

Contents

2 Getting Started

3 Defining an Active Standby Pair Replication Scheme

4 Administering an Active Standby Pair Without Cache Groups

5 Administering an Active Standby Pair with Cache Groups

6 Altering an Active Standby Pair

7 Using Oracle Clusterware to Manage Active Standby Pairs

8 TimesTen Configuration Attributes for Oracle Clusterware

9 Defining Replication Schemes

10 Setting Up a Replicated System

11 Managing Database Failover and Recovery

12 Monitoring Replication

13 Altering Replication

14 Resolving Replication Conflicts

Index

Preface

Oracle TimesTen In-Memory Database is a memory-optimized relational database. Deployed in the application tier, Oracle TimesTen In-Memory Database operates on databases that fit entirely in physical memory using standard SQL interfaces. High availability for the in-memory database is provided through real-time transactional replication.

Audience

This document is intended for application developers and system administrators who use and administer TimesTen to TimesTen Replication.

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 (JavaDataBase Connectivity).

Related documents

TimesTen documentation is available on the product distribution media and on the Oracle Technology Network:

http://www.oracle.com/technetwork/products/timesten/documentation

Conventions

TimesTen supports multiple platforms. Unless otherwise indicated, the information in this guide applies to all supported platforms. The term Windows refers to all supported Windows platforms. The term UNIX applies to all supported UNIX and Linux platforms. See "Platforms" in *Oracle TimesTen In-Memory Database Release Notes* for specific platform versions supported by TimesTen.

> **Note:** In TimesTen documentation, the terms "data store" and "database" are equivalent. Both terms refer to the TimesTen database unless otherwise noted.

This document uses the following text conventions:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.

Convention	Meaning
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.
`italic monospace`	Italic monospace type indicates 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.
[ ]	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 alternative arguments.
. . .	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:

Convention	Meaning
`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.
`bits` or *`bb`*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
`release` or *`rr`*	The first three parts in a release number, with or without dots. The first three parts of a release number represent a major TimesTen release. For example, 1122 or 11.2.2 represents TimesTen 11*g* Release 2 (11.2.2).
`jdk_version`	Two digits that represent the version number of the major JDK release. Specifically, 14 represent JDK 1.4; 5 represents JDK 5.
`DSN`	The data source name.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

What's New

This preface summarizes the new features of Oracle TimesTen In-Memory Database release 11.2.2 that are documented in this guide. It provides links to more information.

New features in Release 11.2.2.4.0

You can now specify an alias or the IP address of the network interface when you want to use a specific local or remote network interface over which database duplication occurs. For details, see "Duplicating a database" on page 4-2.

New features in Release 11.2.2.2.0

- By default, replication is performed with a single thread. You can increase your performance by configuring parallel replication, which configures multiple threads for sending updates from the source database to the target database and for applying the updates on the target database.

  There are two types of parallel replication: automatic and user-defined. In this release, automatic parallel replication is introduced. For more information, see "Configuring parallel replication" on page 10-7.

New features in Release 11.2.2.1.0

- You have additional control over TimesTen application behavior when Oracle Clusterware is managing a TimesTen active standby pair. The `AppFailureInterval`, `AppRestartAttempts` and `AppUptimeThreshold` Clusterware attributes are new. See "Implementing application failover" on page 7-6.

New features in Release 11.2.2.0.0

- Durable commit behavior has changed. See "DURABLE COMMIT" on page 3-10 for active standby pairs and "DURABLE COMMIT" on page 9-22 for other replication schemes.
- LOB columns can be replicated. See "Table requirements and restrictions for active standby pairs" on page 3-3 and "Table requirements and restrictions for replication schemes" on page 9-7.
- TimesTen provides in-memory columnar compression. However, you cannot replicate tables with compressed columns. This restriction is mentioned in "Table

requirements and restrictions for active standby pairs" on page 3-3 and "Table requirements and restrictions for replication schemes" on page 9-7.

1

Overview of TimesTen Replication

The following sections provide an overview of TimesTen replication:

What is replication?

Replication is the process of maintaining copies of data in multiple databases. The purpose of replication is to make data highly available to applications with minimal performance impact. TimesTen recommends the *active standby pair* configuration for highest availability. In an active standby pair replication scheme, the data is copied from the active database to the standby database before being copied to read-only subscribers.

In addition to providing recovery from failures, replication schemes can also distribute application workloads across multiple databases for maximum performance and facilitate online upgrades and maintenance.

Replication is the process of copying data from a *master* database to a subscriber database. Replication is controlled by *replication agents* for each database. The replication agent on the master database reads the records from the transaction log for the master database. It forwards changes to replicated elements to the replication agent on the subscriber database. The replication agent on the subscriber database then applies the updates to its database. If the subscriber replication agent is not running when the updates are forwarded by the master, the master retains the updates in its transaction log until they can be applied at the subscriber database.

An entity that is replicated between databases is called a *replication element*. TimesTen supports databases, cache groups, tables and sequences as replication elements. TimesTen also replicates XLA bookmarks. An active standby pair is the only supported replication scheme for databases with cache groups.

Requirements for replication compatibility

TimesTen replication is supported only between identical platforms and bit-levels. Although you can replicate between databases that reside on the same host, replication is generally used for copying updates into a database that resides on another host. This helps prevent data loss from host failure.

The databases must have DSNs with identical `DatabaseCharacterSet` and `TypeMode` database attributes.

Replication agents

Replication between databases is controlled by a replication agent. Each database is identified by:

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

The replication agent on the master database reads the records from the transaction log and forwards any detected changes to replicated elements to the replication agent on the subscriber database. The replication agent on the subscriber database then applies the updates to its database. 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 replication agents communicate through TCP/IP stream sockets. The replication agents obtain the TCP/IP address, host name, and other configuration information from the replication tables described in *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

Copying updates between databases

Updates are copied between databases in asynchronously by default. Asynchronous replication provides the best performance, but it does not provide the application with confirmation that the replicated updates have been committed on the subscriber databases. For applications that need higher levels of confidence that the replicated data is consistent between the master and subscriber databases, you can enable either *return receipt* or *return twosafe* service.

The *return receipt* service loosely 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 these sections:

- Default replication
- Return receipt replication
- Return twosafe replication

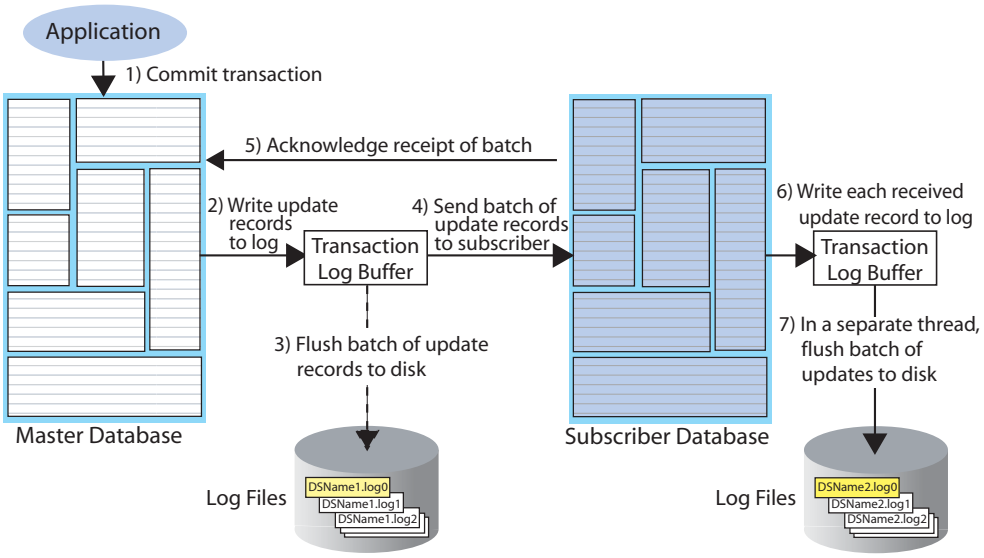
Default replication

When using default TimesTen replication, an application updates a master database and continues working without waiting for the updates to be received and applied by

the subscribers. The master and subscriber databases 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 1–1 Basic asynchronous replication cycle



The default TimesTen replication cycle is:

1. The application commits a local transaction to the master database and is free to continue with other transactions.
2. During the commit, the TimesTen daemon writes the transaction update records to the transaction log buffer.
3. The replication agent on the master database directs the daemon to flush a batch of update records for the committed transactions from the log buffer to a transaction log file. This step ensures that, if the master fails and you need to recover the database from the checkpoint and transaction log files, the recovered master contains all the data it replicated to the subscriber.
4. The master replication agent forwards the batch of transaction update records to the subscriber replication agent, which applies them to the subscriber database. Update records are flushed to disk and forwarded to the subscriber in batches of 256K or less, depending on the master database'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.
5. The subscriber replication agent sends an acknowledgement back to the master replication agent that the batch of update records was received. The 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 database and directs the daemon to write the transaction update records to the transaction log buffer.
7. The replication agent at the subscriber database uses a separate thread to direct the daemon to flush the update records to a transaction log file.

Return receipt replication

The return receipt service provides a level of synchronization between the master and a subscriber database 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 database in the same manner as in the basic asynchronous case. However, when the application commits a transaction that updates a replicated element, the master database 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 applications with the ability to ensure higher levels of data integrity and consistency between the master and subscriber databases. 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 database.

Figure 1–2 Return receipt replication

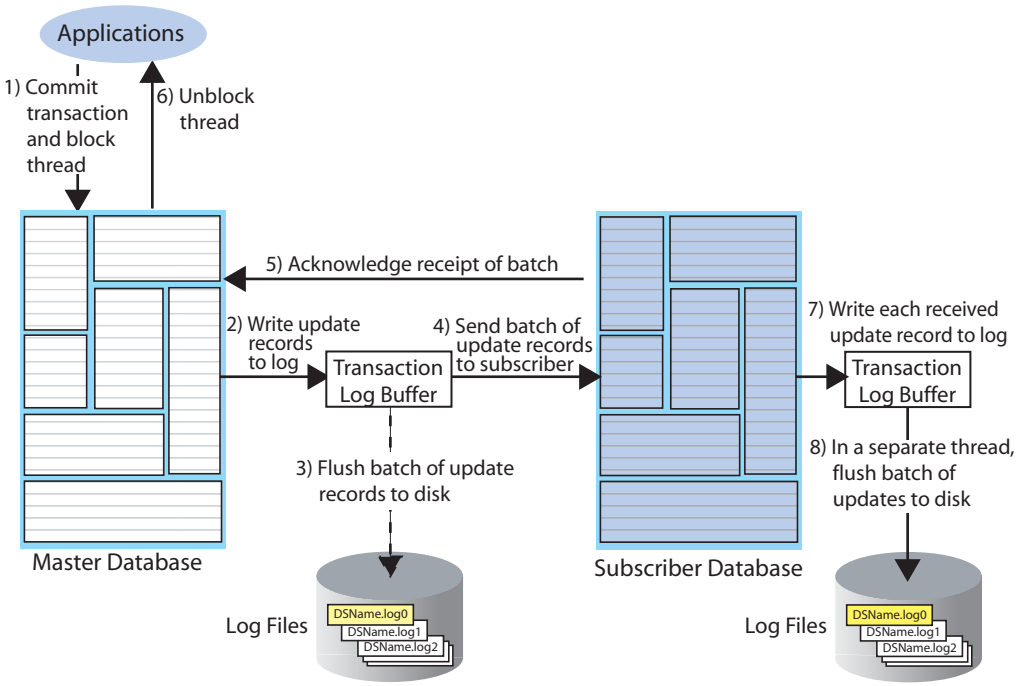


Figure 1–2 shows that the return receipt replication cycle is the same as shown for the basic asynchronous cycle in Figure 1–1, only the master replication agent blocks the application thread after it commits a transaction (Step 1) and retains control of the thread until the subscriber acknowledges receipt 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.

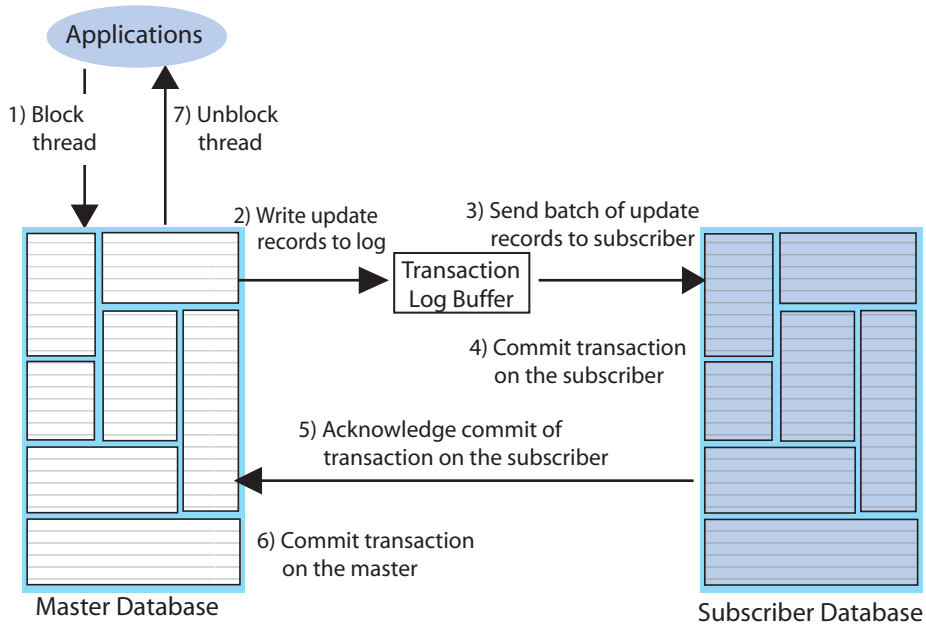
If the subscriber is unable to acknowledge receipt of the transaction within a configurable timeout period (default is 10 seconds), the master replication agent returns a warning stating that it did not receive acknowledgement of the update from the subscriber and returns control of the thread to the 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 time out 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 9-18.

See "RETURN RECEIPT" on page 9-13 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.

Figure 1–3 Return twosafe replication



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 database.
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 is 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 database.
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 database.
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 database. 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 9-15 for information on how to configure replication for return twosafe.

Types of replication schemes

You create a replication scheme to define a specific configuration of master and subscriber databases. This section describes the possible relationships you can define between master and subscriber databases when creating a replication scheme.

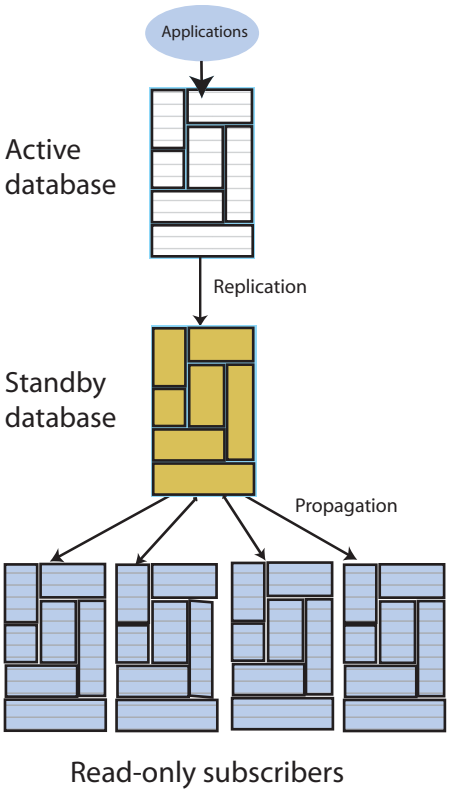
When defining a relationship between a master and subscriber, consider these replication schemes:

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

Active standby pair with read-only subscribers

Figure 1–4 shows an active standby pair replication scheme with an active database, a standby database, and four read-only subscriber databases.

Figure 1–4 Active standby pair



The active standby pair can replicate a whole database or select elements like tables and cache groups.

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

Only one of the master databases can function as an active database at a specific time. You can manage failover and recovery of an active standby pair with Oracle Clusterware. See Chapter 7, "Using Oracle Clusterware to Manage Active Standby Pairs". You can also manage failover and recovery manually. See Chapter 4, "Administering an Active Standby Pair Without Cache Groups".

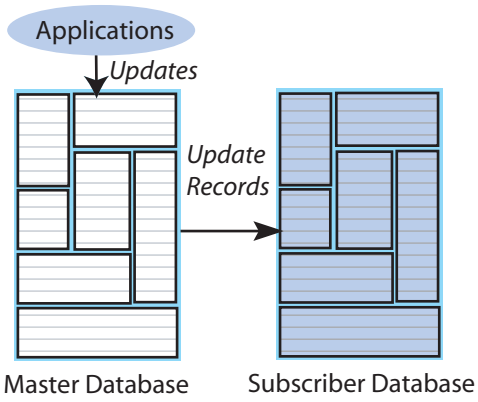
If the standby database fails, the active database can replicate changes directly to the read-only subscribers. After the standby database has been recovered, it contacts the active database to receive any updates that have been sent to the subscribers while the standby was down or was recovering. When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

For details about setting up an active standby pair, see "Setting up an active standby pair with no cache groups" on page 4-3.

Full database replication or selective replication

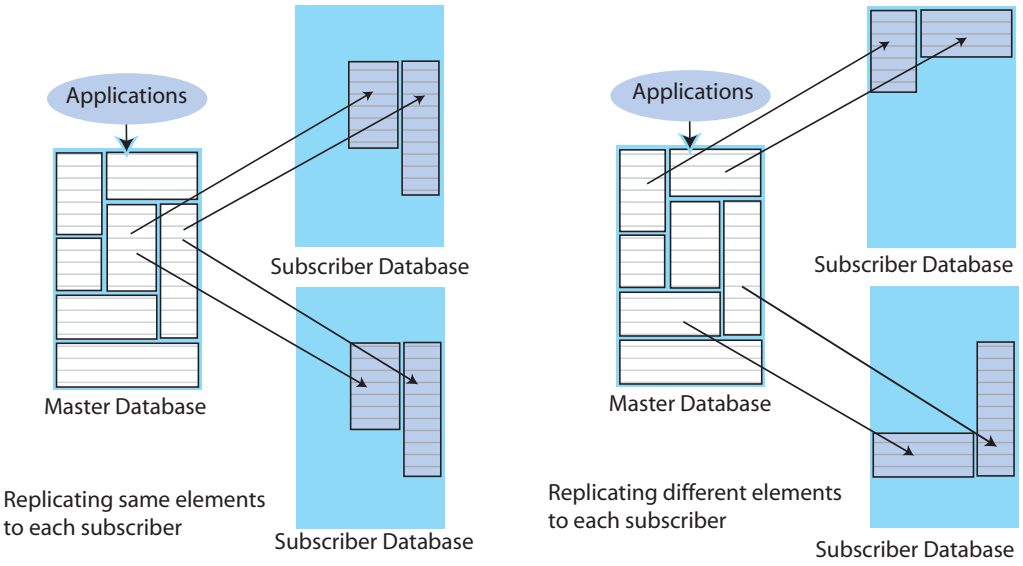
Figure 1–5 illustrates a full replication scheme in which the entire master database is replicated to the subscriber.

Figure 1–5 Replicating the entire master database



You can also configure your master and subscriber databases to selectively replicate some elements in a master database to subscribers. Figure 1–6 shows examples of selective replication. The left side of the figure shows a master database that replicates the same selected elements to multiple subscribers, while the right side shows a master that replicates different elements to each subscriber.

Figure 1–6 Replicating selected elements to multiple subscribers



Unidirectional or bidirectional replication

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

These are basic ways to use bidirectional replication:

- Split workload configuration

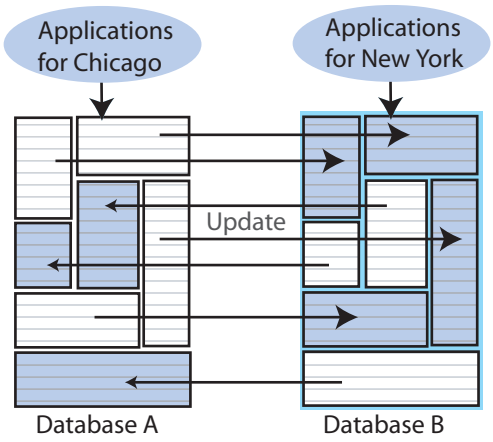
- Distributed workload

Split workload configuration

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

Consider the example shown in Figure 1–7, where the accounts for Chicago are processed on database A while the accounts for New York are processed on database B.

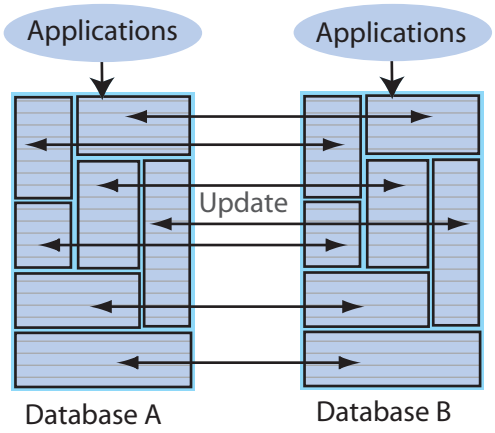
Figure 1–7 Split workload bidirectional replication



Distributed workload

In a distributed workload replication scheme, user access is distributed across duplicate application/database 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/database combination.The distributed workload configuration is shown in Figure 1–8. Users access duplicate applications on each database, which serves as both master and subscriber for the other database.

Figure 1–8 Distributed workload configuration



When databases 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 Chapter 14, "Resolving Replication Conflicts".

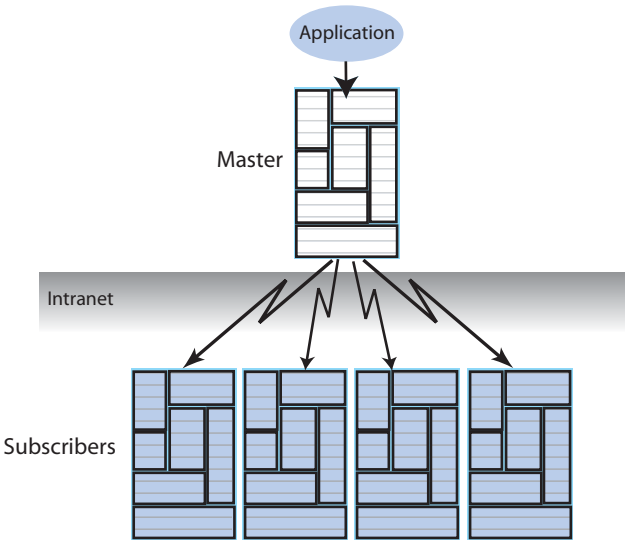
Note: Do not use a distributed workload configuration with the return twosafe return service.

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.

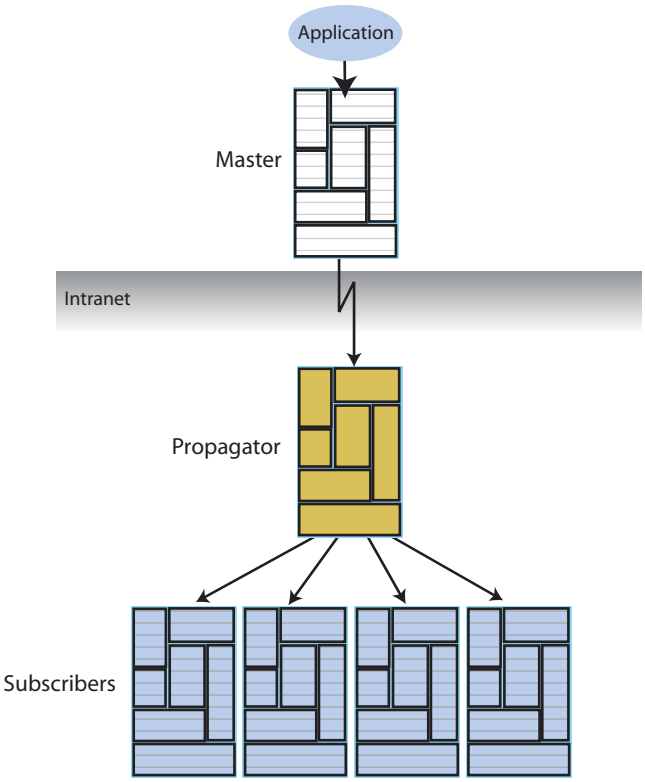
Propagators are useful for optimizing replication performance over lower-bandwidth network connections, such as those between servers in an intranet. For example, consider the direct replication configuration illustrated in Figure 1–9, where a master directly replicates to four subscribers over an intranet connection. Replicating to each subscriber over a network connection in this manner is an inefficient use of network bandwidth.

Figure 1–9 Master replicating directly to multiple subscribers over a network



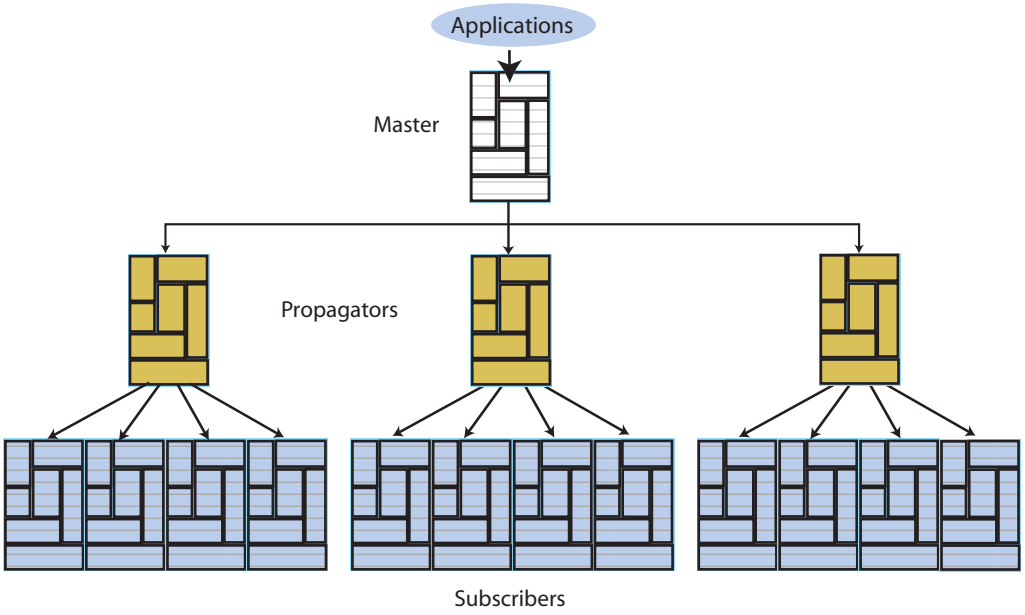
For optimum performance, consider the configuration shown in Figure 1–10, where the master replicates to a single propagator over the network connection. The propagator in turn forwards the updates to each subscriber on its local area network.

Figure 1–10 Master replicating to a single propagator over a network



Propagators are also useful for distributing replication loads in configurations that involve a master database 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 1–11.

Figure 1–11 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.

Cache groups and replication

As described in *Oracle In-Memory Database Cache User's Guide*, a cache group is a group of tables stored in a central Oracle database that are cached in a local Oracle In-Memory Database Cache (IMDB Cache). This section describes how cache groups can be replicated between TimesTen databases. You can achieve high availability by using an active standby pair to replicate asynchronous writethrough cache groups or read-only cache groups.

This section describes the following ways to replicate cache groups:

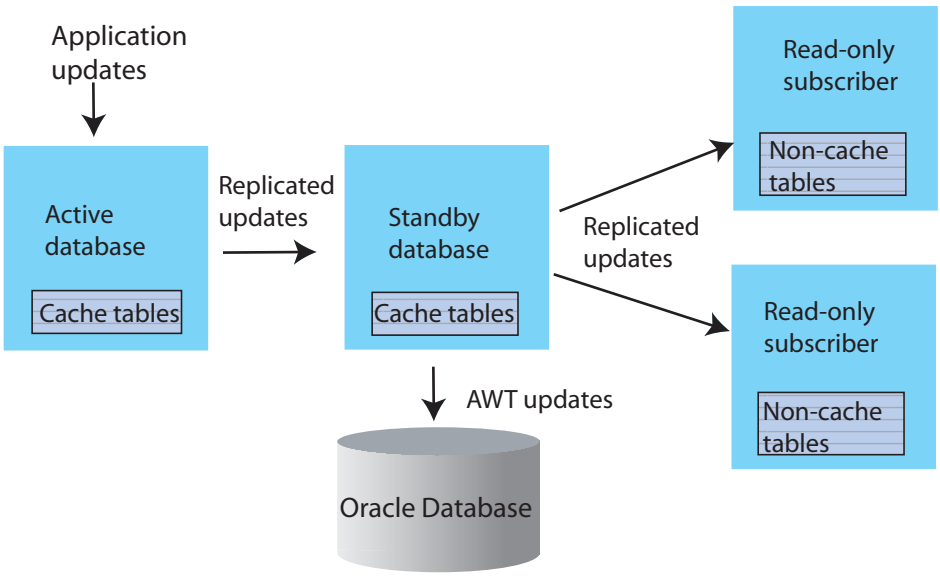
- Replicating an AWT cache group
- Replicating an AWT cache group with a subscriber propagating to an Oracle database
- Replicating a read-only cache group

See Chapter 5, "Administering an Active Standby Pair with Cache Groups" for details about configuring replication of cache groups.

Replicating an AWT cache group

An asynchronous writethrough (AWT) cache group can be configured as part of an active standby pair with optional read-only subscribers to ensure high availability and to distribute the application workload. Figure 1–12 shows this configuration.

Figure 1–12 AWT cache group replicated by an active standby pair



Application updates are made to the active database, the updates are replicated to the standby database, and then the updates are asynchronously written to the Oracle database by the standby. At the same time, the updates are also replicated from the

standby to the read-only subscribers, which may be used to distribute the load from reading applications. The tables on the read-only subscribers are not in cache groups.

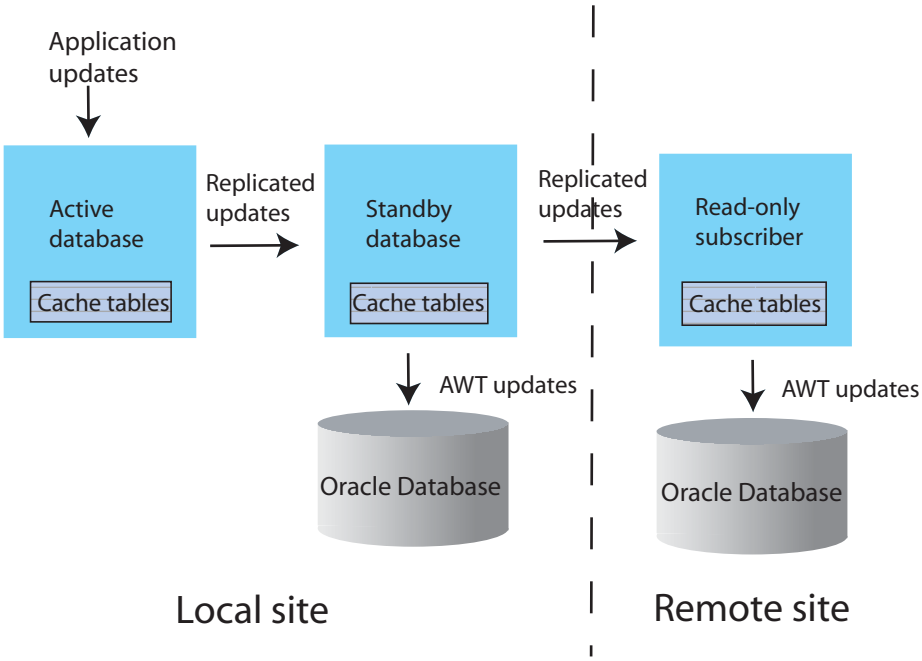
When there is no standby database, the active both accepts application updates and writes the updates asynchronously to the Oracle database and the read-only subscribers. This situation can occur when the standby has not yet been created, or when the active fails and the standby becomes the new active. TimesTen reconfigures the AWT cache group when the standby becomes the new active.

If a failure occurs on the node where the active database resides, the standby node becomes the new active node. TimesTen automatically reconfigures the AWT cache group so that it can be updated directly by the application and continue to propagate the updates to Oracle asynchronously.

Replicating an AWT cache group with a subscriber propagating to an Oracle database

You can recover from a complete failure of a site by creating a special disaster recovery read-only subscriber on a remote site as part of the active standby pair replication configuration. Figure 1–13 shows this configuration.

Figure 1–13 Disaster recovery configuration with active standby pair

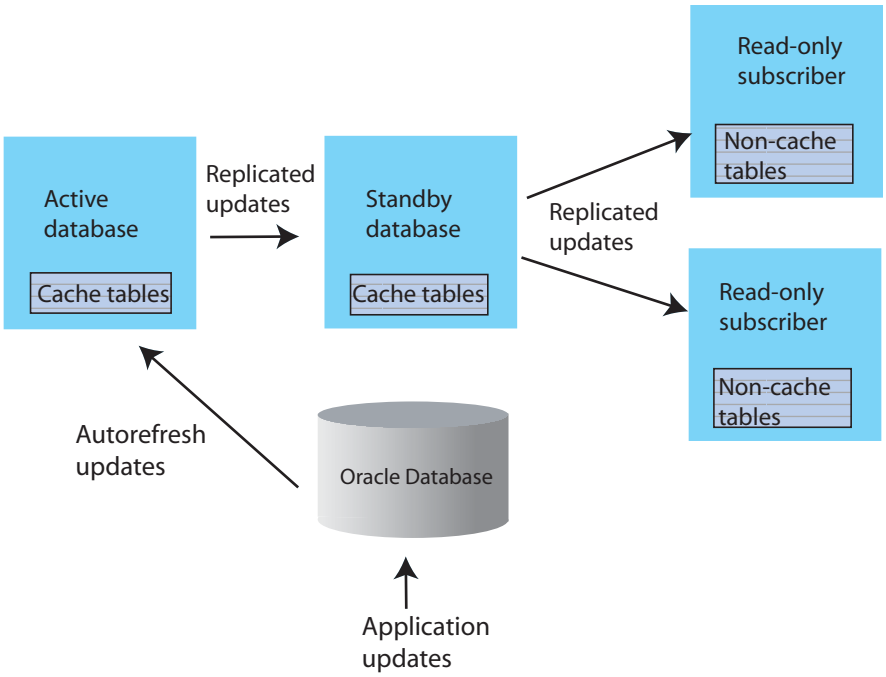


The standby database sends updates to cache group tables on the read-only subscriber. This special subscriber is located at a remote disaster recovery site and can propagate updates to a second Oracle database, also located at the disaster recovery site. You can set up more than one disaster recovery site with read-only subscribers and Oracle databases. See "Using a disaster recovery subscriber in an active standby pair" on page 5-9.

Replicating a read-only cache group

A read-only cache group enforces caching behavior in which committed updates on the Oracle tables are automatically refreshed to the corresponding TimesTen cache tables. Figure 1–14 shows a read-only cache group replicated by an active standby pair.

Figure 1–14 Read-only cache group replicated by an active standby pair



When the read-only cache group is replicated by an active standby pair, the cache group on the active database is autorefreshed from the Oracle database and replicates the updates to the standby, where `AUTOREFRESH` is also configured on the cache group but is in the `PAUSED` state. In the event of a failure of the active, TimesTen automatically reconfigures the standby to be autorefreshed when it takes over for the failed master database by setting the `AUTOREFRESH STATE` to `ON`.

TimesTen also tracks whether updates that have been autorefreshed from the Oracle database to the active database have been replicated to the standby. This ensures that the autorefresh process picks up from the correct point after the active fails, and no autorefreshed updates are lost.

This configuration may also include read-only subscriber databases.This allows the read workload to be distributed across many databases. The cache groups on the standby database replicate to regular (non-cache) tables on the subscribers.

Sequences and replication

In some replication configurations, you may need to keep sequences synchronized between two or more databases. For example, you may have a master database containing a replicated table that uses a sequence to fill in the primary key value for each row. The subscriber database is used as a hot backup for the master database. 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 incremented sequence value to be replicated to subscriber databases, ensuring that rows in this configuration inserted on either database does not conflict. See "Replicating sequences" on page 9-9 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 database.

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 database, 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 database.

Aging and replication

When a table or cache group is configured with least recently used (LRU) or time-based aging, the following rules apply to the interaction with replication:

- The aging configuration on replicated tables and cache groups must be identical on every peer database.
- If the replication scheme is an active standby pair, then aging is performed only on the active database. Deletes that result from aging are then replicated to the standby database. The aging configuration must be set to `ON` on both the active and standby databases. TimesTen automatically determines which database is actually performing the aging based on its current role as active or standby.
- In a replication scheme that is not an active standby pair, aging is performed individually in each database. Deletes performed by aging are not replicated to other databases.
- When an asynchronous writethrough cache group is in a database that is replicated by an active standby pair, delete operations that result from aging are not propagated to the Oracle database.

2

Getting Started

This chapter describes how to configure and start up sample replication schemes. It includes these topics:

- Configuring an active standby pair with one subscriber
- Configuring a replication scheme with one master and one subscriber

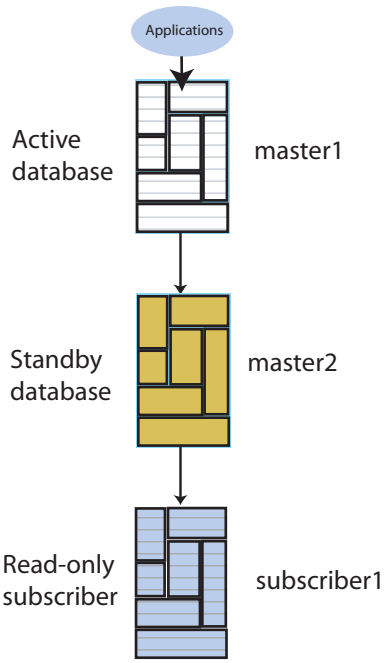
You must have the `ADMIN` privilege to complete the procedures in this chapter.

Configuring an active standby pair with one subscriber

This section describes how to create an active standby pair with one subscriber. The active database is `master1`. The standby database is `master2`. The subscriber database is `subscriber1`. To keep the example simple, all databases reside on the same computer, `server1`.

Figure 2–1 shows this configuration.

Figure 2–1 Active standby pair with one subscriber



This section includes the following topics:

Step 1: Create the DSNs for the master and the subscriber databases

Create DSNs named `master1`, `master2` and `subscriber1` as described in "Managing TimesTen Databases" in *Oracle TimesTen In-Memory Database Operations Guide*.

On UNIX systems, use a text editor to create the following `odbc.ini` file:

```
[master1]
DRIVER=install_dir/lib/libtten.so
DataStore=/tmp/master1
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
[master2]
DRIVER=install_dir/lib/libtten.so
DataStore=/tmp/master2
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
[subscriber1]
DRIVER=install_dir/lib/libtten.so
DataStore=/tmp/subscriber1
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

On Windows, use the ODBC Administrator to set the same connection attributes. Use defaults for all other settings.

Step 2: Create a table in one of the master databases

Use the `ttIsql` utility to connect to the `master1` database:

```
% ttIsql master1

Copyright (c) 1996-2011, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=master1";
Connection successful: DSN=master1;UID=terry;DataStore=/tmp/master1;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;TypeMode=0;
(Default setting AutoCommit=1)
```

```
Command>
```

Create a table called `tab` with columns `a` and `b`:

```
Command> CREATE TABLE tab (a NUMBER NOT NULL,
       > b CHAR(18),
       > PRIMARY KEY (a));
```

Step 3: Define the active standby pair

Define the active standby pair on `master1`:

```
Command> CREATE ACTIVE STANDBY PAIR master1, master2
       > SUBSCRIBER subscriber1;
```

For more information about defining an active standby pair, see Chapter 3, "Defining an Active Standby Pair Replication Scheme".

Step 4: Start the replication agent on a master database

Start the replication agent on `master1`:

```
Command> CALL ttRepStart;
```

Step 5: Set the state of a master database to 'ACTIVE'

The state of a new database in an active standby pair is `'IDLE'` until the active database has been set.

Use the `ttRepStateSet` built-in procedure to designate `master1` as the active database:

```
Command> CALL ttRepStateSet('ACTIVE');
```

Verify the state of `master1`:

```
Command> CALL ttRepStateGet;
< ACTIVE, NO GRID >
1 row found.
```

Step 6. Create a user on the active database

Create a user `terry` with a password of `terry` and grant `terry` the `ADMIN` privilege. Creating a user with the `ADMIN` privilege is required by Access Control for the next step.

```
Command> CREATE USER terry IDENTIFIED BY terry;
User created.
Command> GRANT admin TO terry;
```

Step 7: Duplicate the active database to the standby database

Exit `ttIsql` and use the `ttRepAdmin` utility with the `-duplicate` option to duplicate the active database to the standby database. If you are using two different hosts, enter the `ttRepAdmin` command from the target host.

```
% ttRepAdmin -duplicate -from master1 -host server1 -uid terry -pwd terry
"dsn=master2"
```

Step 8: Start the replication agent on the standby database

Use `ttIsql` to connect to `master2` and start the replication agent:

```
% ttIsql master2
Copyright (c) 1996-2011, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=master2";
Connection successful: DSN=master2;UID=terry;DataStore=/tmp/master2;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;TypeMode=0;
(Default setting AutoCommit=1)
Command> CALL ttRepStart;
```

Starting the replication agent for the standby database automatically sets its state to `'STANDBY'`. Verify the state of `master2`:

```
Command> CALL ttRepStateGet;
< STANDBY, NO GRID >
1 row found.
```

Step 9. Duplicate the standby database to the subscriber

Use the `ttRepAdmin` utility to duplicate the standby database to the subscriber database:

```
% ttRepAdmin -duplicate -from master2 -host server1 -uid terry -pwd terry
"dsn=subscriber1"
```

Step 10: Start the replication agent on the subscriber

Use `ttIsql` to connect to `subscriber1` and start the replication agent. Verify the state of `subscriber1`.

```
% ttIsql subscriber1

Copyright (c) 1996-2011, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=subscriber1";
Connection successful: DSN=subscriber1;UID=terry;DataStore=/stmp/subscriber1;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;TypeMode=0;
(Default setting AutoCommit=1)
Command> CALL ttRepStart;
Command> call ttRepStateGet;
< IDLE, NO GRID >
1 row found.
```

Step 11: Insert data into the table on the active database

Insert a row into the `tab` table on `master1`.

```
Command> INSERT INTO tab VALUES (1,'Hello');
1 row inserted.
Command> SELECT * FROM tab;
< 1, Hello             >
1 row found.
```

Verify that the insert is replicated to `master2` and `subscriber1`.

```
Command> SELECT * FROM tab;
< 1, Hello             >
```

```
1 row found.
```

Step 12: Drop the active standby pair and the table

Stop the replication agents on each database:

```
Command> CALL ttRepStop;
```

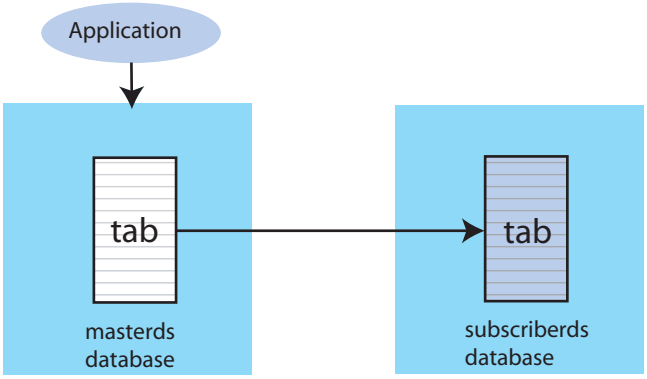
Drop the active standby pair on each database. You can then drop the table `tab` on any database in which you have dropped the active standby pair.

```
Command> DROP ACTIVE STANDBY PAIR;
Command> DROP TABLE tab;
```

Configuring a replication scheme with one master and one subscriber

This section describes how to configure a replication scheme that replicates the contents of a single table in a master database (`masterds`) to a table in a subscriber database (`subscriberds`). To keep the example simple, both databases reside on the same computer.

Figure 2–2 Simple replication scheme



This section includes the following topics:

Step 1: Create the DSNs for the master and the subscriber

Create DSNs named `masterds` and `subscriberds` as described in "Managing TimesTen Databases" in *Oracle TimesTen In-Memory Database Operations Guide*.

On UNIX systems, use a text editor to create the following `odbc.ini` file on each database:

```
[masterds]
DataStore=/tmp/masterds
DatabaseCharacterSet=AL32UTF8
```

```
ConnectionCharacterSet=AL32UTF8
[subscriberds]
DataStore=/tmp/subscriberds
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

On Windows, use the ODBC Administrator to set the same connection attributes. Use defaults for all other settings.

Step 2: Create a table and replication scheme on the master database

Connect to `masterds` with the `ttIsql` utility:

```
% ttIsql masterds
Copyright (c) 1996-2011, Oracle.  All rights reserved.
Type ? or "help" for help, type "exit" to quit ttIsql.

connect "DSN=masterds";
Connection successful: DSN=masterds;UID=ttuser;
DataStore=/tmp/masterds;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;TypeMode=0;
(Default setting AutoCommit=1)
Command>
```

Create a table named `tab` with columns named `a`, `b` and `c`:

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

Create a replication scheme called `repscheme` to replicate the `tab` table from `masterds` to `subscriberds`.

```
Command> CREATE REPLICATION repscheme
       > ELEMENT e TABLE tab
       > MASTER masterds
       > SUBSCRIBER subscriberds;
```

Step 3: Create a table and replication scheme on the subscriber database

Connect to `subscriberds` and create the same table and replication scheme, using the same procedure described in Step 2.

Step 4: Start the replication agent on each database

Start the replication agents on `masterds` and `subscriberds`:

```
Command> call ttRepStart;
```

Exit `ttIsql`. Use the `ttStatus` utility to verify that the replication agents are running for both databases:

```
% ttStatus
TimesTen status report as of Thu Aug 11 17:05:23 2011

Daemon pid 18373 port 4134 instance ttuser
TimesTen server pid 18381 started on port 4136
------------------------------------------------------------------------
Data store /tmp/masterds
There are 16 connections to the data store
```

```
Shared Memory KEY 0x0201ab43 ID 5242889
PL/SQL Memory KEY 0x0301ab43 ID 5275658 Address 0x10000000
Type            PID     Context     Connection Name              ConnID
Process         20564   0x081338c0  masterds                          1
Replication     20676   0x08996738  LOGFORCE                          5
Replication     20676   0x089b69a0  REPHOLD                           2
Replication     20676   0x08a11a58  FAILOVER                          3
Replication     20676   0x08a7cd70  REPLISTENER                       4
Replication     20676   0x08ad7e28  TRANSMITTER                       6
Subdaemon       18379   0x080a11f0  Manager                        2032
Subdaemon       18379   0x080fe258  Rollback                       2033
Subdaemon       18379   0x081cb818  Checkpoint                     2036
Subdaemon       18379   0x081e6940  Log Marker                     2035
Subdaemon       18379   0x08261e70  Deadlock Detector              2038
Subdaemon       18379   0xae100470  AsyncMV                        2040
Subdaemon       18379   0xae11b508  HistGC                         2041
Subdaemon       18379   0xae300470  Aging                          2039
Subdaemon       18379   0xae500470  Flusher                        2034
Subdaemon       18379   0xae55b738  Monitor                        2037
Replication policy  : Manual
Replication agent is running.
Cache Agent policy  : Manual
PL/SQL enabled.
------------------------------------------------------------------------
Data store /tmp/subscriberds
There are 16 connections to the data store
Shared Memory KEY 0x0201ab41 ID 5177351
PL/SQL Memory KEY 0x0301ab41 ID 5210120 Address 0x10000000
Type            PID     Context     Connection Name              ConnID
Process         20594   0x081338f8  subscriberds                      1
Replication     20691   0x0893c550  LOGFORCE                          5
Replication     20691   0x089b6978  REPHOLD                           2
Replication     20691   0x08a11a30  FAILOVER                          3
Replication     20691   0x08a6cae8  REPLISTENER                       4
Replication     20691   0x08ad7ba8  RECEIVER                          6
Subdaemon       18376   0x080b1450  Manager                        2032
Subdaemon       18376   0x0810e4a8  Rollback                       2033
Subdaemon       18376   0x081cb8b0  Flusher                        2034
Subdaemon       18376   0x08246de0  Monitor                        2035
Subdaemon       18376   0x082a20a8  Deadlock Detector              2036
Subdaemon       18376   0x082fd370  Checkpoint                     2037
Subdaemon       18376   0x08358638  Aging                          2038
Subdaemon       18376   0x083b3900  Log Marker                     2040
Subdaemon       18376   0x083ce998  AsyncMV                        2039
Subdaemon       18376   0x08469e90  HistGC                         2041
Replication policy  : Manual
Replication agent is running.
Cache Agent policy  : Manual
PL/SQL enabled.
```

Step 5: Insert data into the table on the master database

Use `ttIsql` to connect to the master database and insert some rows into the `tab` table:

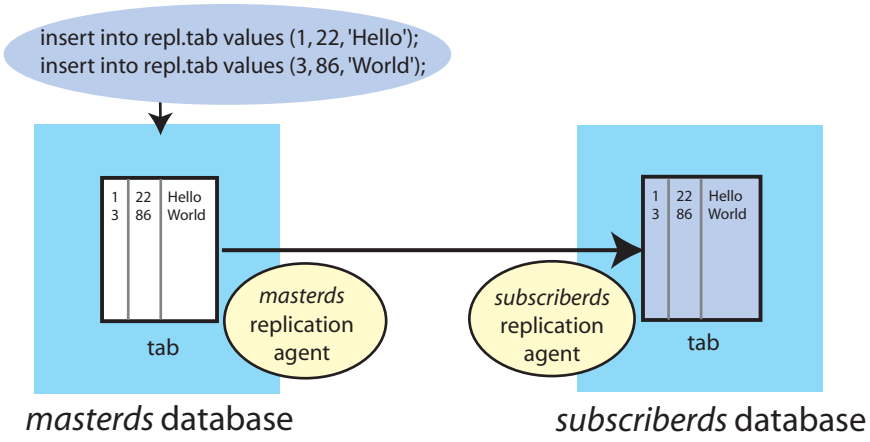
```
% ttIsql masterds
Command> INSERT INTO tab VALUES (1, 22, 'Hello');
1 row inserted.
Command> INSERT INTO tab VALUES (3, 86, 'World');
1 row inserted.
```

Open a second command prompt window for the subscriber. Connect to the subscriber database and check the contents of the `tab` table:

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

Figure 2–3 shows that the rows that are inserted into `masterds` are replicated to `subscriberds`.

Figure 2–3 Replicating changes to the subscriber database



Step 6: Drop the replication scheme and table

After you have completed your replication tests, stop the replication agents on both `masterds` and `subscriberds`:

```
Command> CALL ttRepStop;
```

To remove the `tab` table and `repscheme` replication scheme from the master and subscriber databases, enter these statements on each database:

```
Command> DROP REPLICATION repscheme;
Command> DROP TABLE tab;
```

3

Defining an Active Standby Pair Replication Scheme

The following sections describe how to design a highly available system and define replication schemes:

- Restrictions on active standby pairs
- Defining the DSNs for the databases
- Defining an active standby pair replication scheme
- Identifying the databases in the active standby pair
- Using a return service
- Setting STORE attributes
- Configuring network operations
- Using automatic client failover for an active standby pair
- Including or excluding database objects from replication
- Materialized views in an active standby pair
- Replicating sequences in an active standby pair

To reduce the amount of bandwidth required for replication, see "Compressing replicated traffic" on page 3-11.

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 databases.
- Each master and subscriber database must be on a different node to ensure high availability.
- The active database and the standby database should be on the same LAN.
- The clock skew between the active node and the standby node cannot exceed 250 milliseconds.
- For the initial set-up, you can create a standby database only by duplicating the active database with the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function.

- Read-only subscribers can be created only by duplicating the standby database. If the standby database is unavailable, then the read-only subscribers can be created by duplicating the active database. See "Duplicating a database" on page 4-2.
- After failover, the new standby database can only be recovered from the active database by duplicating the active database *unless* return twosafe replication is used between the active and the standby databases. If return twosafe replication is used, the automated master catch-up feature may be used instead. See "Automatic catch-up of a failed master database" on page 11-3.
- Writes on replicated tables are not allowed on the standby database and the subscriber databases. However, operations on sequences and XLA bookmarks *are* allowed on the standby database and the subscriber databases. Reads are also allowed.
- Replication from the standby database to the read-only subscribers occurs asynchronously.
- `ALTER ACTIVE STANDBY PAIR` statements can be executed only on the active database. If `ALTER ACTIVE STANDBY PAIR` is executed on the active database, then the standby database must be regenerated by duplicating the active database. All subscribers must also be regenerated from the standby database. See "Duplicating a database" on page 4-2.
- You cannot replicate a temporary database.
- You cannot replicate tables with compressed columns.

Defining the DSNs for the databases

Before you define the active standby pair, define the DSNs for the active, standby and read-only subscriber databases. On UNIX, create an `odbc.ini` file. On Windows, use the ODBC Administrator to name the databases and set connection attributes. See "Step 1: Create the DSNs for the master and the subscriber databases" on page 2-2 for an example.

Each database "name" specified in a replication scheme must match the prefix of the database file name without the path given for the `DataStore` data store attribute in the DSN definition for the database. To avoid confusion, use the same name for both the `DataStore` and `Data Source Name` data store attributes in each DSN definition. Values for `DataStore` are case-sensitive. If the database path is *`directory/subdirectory`*`/foo.ds0`, then `foo` is the database name that you should use.

Defining an active standby pair replication scheme

Use the `CREATE ACTIVE STANDBY PAIR` SQL statement to create an active standby pair replication scheme. The complete syntax for the `CREATE ACTIVE STANDBY PAIR` statement is provided in the *Oracle TimesTen In-Memory Database SQL Reference*.

You must have the `ADMIN` privilege to use the `CREATE ACTIVE STANDBY PAIR` statement and to perform other replication operations. Only the instance administrator can duplicate databases.

Table 3–1 shows the components of an active standby pair replication scheme and identifies the parameters associated with the topics in this chapter.

Table 3–1 Components of an active standby pair replication scheme

Component	See...
`CREATE ACTIVE STANDBY PAIR` *`FullDatabaseName, FullDatabaseName`*	"Identifying the databases in the active standby pair" on page 3-3
`[`*`ReturnServiceAttribute`*`]`	"Using a return service" on page 3-4
`[SUBSCRIBER` *`FullDatabaseName`* `[,...]]`	"Identifying the databases in the active standby pair" on page 3-3
`[STORE` *`FullDatabaseName`* `[`*`StoreAttribute`* `[...]]]`	"Setting STORE attributes" on page 3-6
`[`*`NetworkOperation`* `[...]]`	"Configuring network operations" on page 3-12
`[{INCLUDE|EXCLUDE}` `{TABLE [[`*`Owner`*`.]`*`TableName`*`[,...]]|` `CACHE GROUP [[`*`Owner`*`.]`*`CacheGroupName`*`[,...]|` `SEQUENCE [[`*`Owner`*`.]`*`SequenceName`*`[,...]]}` `[,...]]`	"Including or excluding database objects from replication" on page 3-13

Identifying the databases in the active standby pair

Use the full database name described in "Defining the DSNs for the databases" on page 3-2. The first database name designates the active database. The second database name designates the standby database. Read-only subscriber databases are indicated by the `SUBSCRIBER` clause.

You can also specify the hosts where the databases reside by using an IP address or a literal host name surrounded by double quotes.

The active database and the standby database should be on separate hosts to achieve a highly available system. Read-only subscribers can be either local or remote. A remote subscriber provides protection from site-specific disasters.

Provide a host ID as part of *`FullDatabaseName`*:

```
DatabaseName [ON Host]
```

`Host` can be either an IP address or a literal host name. Use the value returned by the `hostname` operating system command. It is good practice to surround a host name with double quotes. For example:

```
CREATE ACTIVE STANDBY PAIR
  repdb1_1122 ON "host1",
  repdb2_1122 ON "host2";
```

Table requirements and restrictions for active standby pairs

Tables that are replicated in an active standby pair 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 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 active database must also exist on its counterpart table in the standby database.

> **Note:** The keys on replicated tables are transmitted in each update record to the subscribers. Smaller keys are transmitted more efficiently.

Replicated tables have these data type restrictions:

- VARCHAR2, NVARCHAR2, VARBINARY and TT_VARCHAR columns in replicated tables are limited to a size of 4 megabytes. 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 database character set. For example, if the character set requires four bytes for each character, the maximum possible length is one million characters. For an NVARCHAR2 column, which requires two bytes for each character, the maximum length when using character length semantics is two million characters.
- Columns with the BLOB data type in replicated tables are limited to a size of 16 megabytes. Columns with the CLOB or NCLOB data type in replicated tables are limited to a size of 4 megabytes.
- A primary key column cannot have a LOB data type.

You cannot replicate tables with compressed columns.

Using a return service

You can configure your replication scheme with a return service to ensure a higher level of confidence that your replicated data is consistent on the active and standby databases. See "Copying updates between databases" on page 1-2. This section describes how to configure and manage the return receipt and return twosafe services. NO RETURN (asynchronous replication) is the default and provides the fastest performance.

The following sections describe the following return service clauses:

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

RETURN RECEIPT

TimesTen provides an optional return receipt service to loosely couple or synchronize your application with the replication mechanism.

You can specify the RETURN RECEIPT clause to enable the return receipt service for the standby database. With return receipt enabled, when your application commits a transaction for an element on the active database, the application remains blocked until the standby acknowledges receipt of the transaction update.

If the standby is 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 3-8 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 12-15 for details.

You can also configure the replication agent to disable the return receipt service after a specific number of timeouts. See "Setting the return service timeout period" on page 3-8 for details.

RETURN RECEIPT BY REQUEST

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

If you specify `RETURN RECEIPT BY REQUEST`, you must use the `ttRepSyncSet` built-in 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).

If the standby database is unable to acknowledge receipt of the transaction update within a configurable timeout period, the application receives a `tt_ErrRepReturnFailed` (8170) warning on its commit request. See "Setting the return service timeout period" on page 3-8 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.
```

RETURN TWOSAFE

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 standby database before it is committed on the active database. If replication is unable to verify the transaction has been committed on the standby, it returns notification of the error. Upon receiving an error, the application can either take a unique action or fall back on preconfigured actions, depending on the type of failure.

When replication is configured with `RETURN TWOSAFE`, you must disable autocommit mode.

A transaction that contains operations that are replicated with `RETURN TWOSAFE` cannot have a `PassThrough` setting greater than 0. If `PassThrough` is greater than 0, an error is returned and the transaction must be rolled back.

If the standby is unable to acknowledge commit of the transaction update within a configurable timeout period, the application receives a `tt_ErrRepReturnFailed` (8170) warning on its commit request. See "Setting the return service timeout period" on page 3-8 for more information on the return service timeout period.

RETURN TWOSAFE BY REQUEST

RETURN TWOSAFE enables notification of commit on the standby database for all transactions. You can use the RETURN TWOSAFE BY REQUEST clause to enable notification of a commit on the standby only for specific transactions identified by your application.

A transaction that contains operations that are replicated with RETURN TWOSAFE cannot have a PassThrough setting greater than 0. If PassThrough is greater than 0, an error is returned and the transaction must be rolled back.

If you specify RETURN TWOSAFE BY REQUEST for a standby database, you must use the ttRepSyncSet built-in 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).

If the standby is unable to acknowledge commit of the transaction within the timeout period, the application receives a tt_ErrRepReturnFailed (8170) warning on its commit request. The application can then chose how to handle the timeout, in the same manner as described for "RETURN TWOSAFE" on page 3-5.

The ALTER TABLE statement cannot be used to alter a replicated table that is part of a RETURN TWOSAFE BY REQUEST transaction. If DDLCommitBehavior=0 (the default), the ALTER TABLE operation succeeds because a commit is performed before the ALTER TABLE operation, resulting in the ALTER TABLE operation executing in a new transaction which is not part of the RETURN TWOSAFE BY REQUEST transaction. If DDLCommitBehavior=1, the ALTER TABLE operation results in error 8051.

See "Setting the return service timeout period" on page 3-8 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 clause to explicitly disable either the return receipt or return twosafe service, depending on which one you have enabled. NO RETURN is the default condition.

Setting STORE attributes

Table 3–2 lists the optional STORE attributes for the CREATE ACTIVE STANDBY PAIR statement.

Table 3–2 STORE attribute descriptions

STORE attribute	Description
DISABLE RETURN {SUBSCRIBER|ALL} *NumFailures*	Set the return service policy so that return service blocking is disabled after the number of timeouts specified by *NumFailures*. See "Establishing return service failure/recovery policies" on page 3-9.

Table 3–2 (Cont.) STORE attribute descriptions

STORE attribute	Description
`RETURN SERVICES {ON|OFF} WHEN [REPLICATION] STOPPED`	Set return services on or off when replication is disabled. See "Establishing return service failure/recovery policies" on page 3-9.
`RESUME RETURN` *`Milliseconds`*	If `DISABLE RETURN` has disabled return service blocking, this attribute sets the policy for re-enabling the return service. See "Establishing return service failure/recovery policies" on page 3-9.
`RETURN WAIT TIME` *`Seconds`*	Specifies the number of seconds to wait for return service acknowledgement. A value of 0 means that there is no waiting. The default value is 10 seconds. The application can override this timeout setting by using the `returnWait` parameter in the `ttRepSyncSet` built-in procedure. See "Setting the return service timeout period" on page 3-8.
`DURABLE COMMIT {ON|OFF}`	Overrides the `DurableCommits` general connection attribute setting. `DURABLE COMMIT ON` enables durable commits regardless of whether the replication agent is running or stopped. It also enables durable commits when the `ttRepStateSave` built-in procedure has marked the standby database as failed. See "DURABLE COMMIT" on page 3-10.
`LOCAL COMMIT ACTION {NO ACTION|COMMIT}`	Specifies 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. This is the default. `COMMIT`- On timeout, the commit function attempts to perform a commit to end 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. See "LOCAL COMMIT ACTION" on page 3-10.
`COMPRESS TRAFFIC {ON|OFF}`	Compresses replicated traffic to reduce the amount of network bandwidth used. See "Compressing replicated traffic" on page 3-11.
`PORT` *`PortNumber`*	Sets the port number used by a database to listen for updates from another database. In an active standby pair, the standby database listens for updates from the active database. Read-only subscribers listen for updates from the standby database. 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 "Port assignments" on page 3-11.
`TIMEOUT` *`Seconds`*	Set the maximum number of seconds the replication agent waits for a response from the database.

Table 3–2 (Cont.) STORE attribute descriptions

STORE attribute	Description
FAILTHRESHOLD *Value*	Sets the log failure threshold. See "Setting the log failure threshold" on page 3-11.

The rest of this section includes these topics:

- Setting the return service timeout period
- Compressing replicated traffic
- Port assignments
- Setting the log failure threshold

Setting the return service timeout period

If a replication scheme is configured with one of the return services described in "Using a return service" on page 3-4, a timeout occurs if the standby database is unable to send an acknowledgement back to the active within the time period specified by RETURN WAIT TIME. If the standby database is unable to acknowledge the transaction update from the active database within the timeout period, the 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:

- Specifying the RETURN WAIT TIME in the CREATE ACTIVE STANDBY PAIR statement or ALTER ACTIVE STANDBY PAIR statement. A RETURN WAIT TIME of 0 indicates no waiting.
- Specifying a different return service timeout period programmatically by calling the ttRepSyncSet procedure with a new value for the returnWait parameter. Once set, the timeout period applies to all subsequent return service transactions until you either reset the timeout period or terminate the application session.

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 standby does not necessarily mean the transaction has not been or will not be replicated.

You can respond to return service timeouts by:

- Disabling return service blocking manually
- Establishing return service failure/recovery policies

Disabling return service blocking manually

You may want respond 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 the ALTER ACTIVE STANDBY PAIR statement to disable return receipt blocking. See "Making other changes to an active standby pair" on page 6-5.

- Calling the `ttDurableCommit` built-in procedure to durably commit transactions on the active database that you can no longer verify as being received by the standby

If 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 time out.

Establishing return service failure/recovery policies

An alternative to manually responding to return service timeout failures is to establish return service failure and recovery policies in the 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 a predefined manner.

The following attributes in the `CREATE ACTIVE STANDBY PAIR` statement set the failure and recovery policies when using a `RETURN RECEIPT` or `RETURN TWOSAFE` service:

- RETURN SERVICES {ON | OFF} WHEN [REPLICATION] STOPPED
- DISABLE RETURN
- RESUME RETURN
- DURABLE COMMIT
- LOCAL COMMIT ACTION

The policies set by these attributes are applicable until changed. The replication agent must be running to enforce these policies, with the exception of `DURABLE COMMIT`.

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 active replication agent is stopped (for example, by `ttAdmin -repStop` *`active`*) or the replication state of the standby database is set to `stop` or `pause` with respect to the active database (for example, by `ttRepAdmin -state stop` *`standby`*). A failed standby 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 standby database 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.

DISABLE RETURN When a `DISABLE RETURN` value is set, the database keeps track of the number of return receipt or return twosafe transactions that have exceeded the timeout period set by `RETURN WAIT TIME`. If the number of timeouts exceeds the maximum value set by `DISABLE RETURN`, the application reverts to a default

replication cycle in which it no longer waits for the standby to acknowledge the replicated updates.

Specifying SUBSCRIBER is the same as specifying ALL. Both settings refer to the standby database.

The DISABLE RETURN failure policy is only enabled when the replication agent is running. If DISABLE RETURN is specified without RESUME RETURN, the return services remain off until the replication agent for the database has been restarted. You can cancel this failure policy by stopping the replication agent and specifying DISABLE RETURN 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.

RESUME RETURN When we say return service blocking is "disabled," we mean that the applications on the master database no longer block execution while waiting to receive acknowledgements from the subscribers that they received or committed the replicated updates. Note, however, that the master still listens for an acknowledgement of each batch of replicated updates from the standby database.

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 the standby database, 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 from the subscriber.

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 ACTIVE STANDBY PAIR 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 general connection attribute on the master database and forces durable commits for those transactions that have had return service blocking disabled.

When DURABLE COMMITS are ON, durable commits are issued when return service blocking is disabled regardless of whether the replication agent is running or stopped. They are also issued for an active standby pair in which the ttRepStateSave built-in procedure has marked the standby database as failed.

LOCAL COMMIT ACTION When you are using the return twosafe service, you can specify how the master replication agent responds to timeouts by setting LOCAL COMMIT ACTION. You can override the setting for specific transactions by calling the localAction parameter in the ttRepSyncSet procedure.

The possible actions upon receiving a timeout during replication of a twosafe transaction are:

- COMMIT - On timeout, the commit function attempts to perform a commit to end the transaction locally. No more operations are possible on the same transaction.
- 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. This is the default.

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 database specified by the STORE parameter in the CREATE ACTIVE STANDBY PAIR or ALTER ACTIVE STANDBY PAIR statement. TimesTen does not compress traffic from other databases.

Though the compression algorithm is optimized for speed, enabling the COMPRESS TRAFFIC attribute affects replication throughput and latency.

Example 3–1 Compressing traffic from an active database

For example, to compress replicated traffic from active database dsn1 and leave the replicated traffic from standby database dsn2 uncompressed, the CREATE ACTIVE STANDBY PAIR statement looks like:

```
CREATE ACTIVE STANDBY PAIR dsn1 ON "host1", dsn2 ON "host2"
  SUBSCRIBER dsn3 ON "host3"
  STORE dsn1 ON "host1" COMPRESS TRAFFIC ON;
```

Example 3–2 Compressing traffic from both master databases

To compress the replicated traffic from the dsn1 and dsn2 databases, use:

```
CREATE ACTIVE STANDBY PAIR dsn1 ON "host1", dsn2 ON "host2"
  SUBSCRIBER dsn3 ON "host3"
STORE dsn1 ON "host1" COMPRESS TRAFFIC ON
STORE dsn2 ON "host2" COMPRESS TRAFFIC ON;
```

Port assignments

Static port assignment is recommended. 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.

You must assign static ports if you want to do online upgrades.

When statically assigning ports, it is important to specify the full host name, DSN and port in the STORE attribute of the CREATE ACTIVE STANDBY PAIR statement.

Example 3–3 Assigning static ports

```
CREATE ACTIVE STANDBY PAIR dsn1 ON "host1", dsn2 ON "host2"
  SUBSCRIBER dsn3 ON "host3"
STORE dsn1 ON "host1" PORT 16080
STORE dsn2 ON "host2" PORT 16083
STORE dsn3 ON "host3" PORT 16084;
```

Setting the log failure threshold

You can establish a threshold value that, when exceeded, sets an unavailable standby database or a read-only subscriber to the failed state before the available log space is exhausted.

Set the log threshold by specifying the STORE clause with a FAILTHRESHOLD value in the CREATE ACTIVE STANDBY PAIR or ALTER ACTIVE STANDBY PAIR statement. The default threshold value is 0, which means "no limit."

If an active database sets the standby database or a read-only subscriber to the `failed` state, it drops all of the data for the failed database from its log and transmits a message to the failed database. If the active replication agent can communicate with the replication agent of the failed database, then the message is transmitted immediately. Otherwise, the message is transmitted when the connection is reestablished.

Any application that connects to the failed database receives a `tt_ErrReplicationInvalid` (8025) warning indicating that the database has been marked `failed` by a replication peer. Once the database has been informed of its failed status, its state on the active database is changed from `failed` to `stop`.

An application can use the ODBC `SQLGetInfo` function to check if the database the application is connected to has been set to the `failed` state.

For more information about database states, see Table 10–2, " Database states" on page 10-15.

Configuring network operations

If a 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 specify the database name, you can configure replication to send or receive traffic over a different interface using the `ROUTE` clause.

The syntax of the `ROUTE` clause is:

```
ROUTE MASTER FullDatabaseName SUBSCRIBER FullDatabaseName
  {{MASTERIP MasterHost | SUBSCRIBERIP SubscriberHost}
    PRIORITY Priority} [...]
```

In the context of the `ROUTE` clause, each master database is a subscriber of the other master database and each read-only subscriber is a subscriber of both master databases. This means that the `CREATE ACTIVE STANDBY PAIR` statement should include `ROUTE` clauses in multiples of two to specify a route in both directions. See Example 3–4.

Example 3–4 Configuring multiple network interfaces

If host `host1` is configured with a second interface accessible by the host name `host1fast`, and `host2` 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 ACTIVE STANDBY PAIR dns1, dsn2
ROUTE MASTER dsn1 ON "host1" SUBSCRIBER dsn2 ON "host2"
    MASTERIP "host1fast" PRIORITY 1
    SUBSCRIBERIP "192.168.1.100" PRIORITY 1
ROUTE MASTER dsn2 ON "host2" SUBSCRIBER dsn1 ON "host1"
    MASTERIP "192.168.1.100" PRIORITY 1
    SUBSCRIBERIP "host1fast" 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 can use the `ROUTE` clause to specify two or more interfaces for each master or subscriber that are used by replication in order of priority.

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.

Example 3–5 Configuring network priority

If host host1 is configured with two network interfaces at IP addresses 192.168.1.100 and 192.168.1.101, and host host2 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 ACTIVE STANDBY PAIR dns1, dns2
ROUTE MASTER dsn1 ON "host1" SUBSCRIBER dsn2 ON "host2"
  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;
```

Using automatic client failover for an active standby pair

Automatic client failover is for use in High Availability scenarios with a TimesTen active standby pair replication configuration. If failure of the active TimesTen node results in the original standby node becoming the new active node, then automatic client failover feature automatically transfers the application connection to the new active node.

For full details on how to configure and use automatic client failover, see "Using automatic client failover" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Note: Automatic client failover is complementary to Oracle Clusterware in situations where Oracle Clusterware is used, but the two features are not dependent on each other. For information about Oracle Clusterware, you can refer to Chapter 7, "Using Oracle Clusterware to Manage Active Standby Pairs".

Including or excluding database objects from replication

An active standby pair replicates an entire database by default. Use the INCLUDE clause to replicate *only* the tables, cache groups and sequences that are listed in the INCLUDE clause. No other database objects will be replicated in an active standby pair that is defined with an INCLUDE clause. For example, this INCLUDE clause specifies three tables to be replicated by the active standby pair:

```
INCLUDE TABLE employees, departments, jobs
```

You can choose to exclude specific tables, cache groups or sequences from replication by using the EXCLUDE clause of the CREATE ACTIVE STANDBY PAIR statement. Use one EXCLUDE clause for each object type. For example:

```
EXCLUDE TABLE ttuser.tab1, ttuser.tab2
EXCLUDE CACHE GROUP ttuser.cg1, ttuser.cg2
EXCLUDE SEQUENCE ttuser.seq1, ttuser.seq2
```

Note: Sequences with the `CYCLE` attribute cannot be replicated.

Materialized views in an active standby pair

When you replicate a database containing a materialized or nonmaterialized view, only the detail tables associated with the view are replicated. The view itself is not replicated. A matching view can be defined on the standby database, but it is not required. If detail tables are replicated, TimesTen automatically updates the corresponding view. However, TimesTen replication verifies only that the replicated detail tables have the same structure on both databases. It does not enforce that the materialized views are the same on each database.

Replicating sequences in an active standby pair

Sequences are replicated unless you exclude them from the active standby pair or unless they have the `CYCLE` attribute. See "Including or excluding database objects from replication" on page 3-13. Replication of sequences is optimized by reserving a range of sequence numbers on the standby database each time a sequence is updated on the active database. Reserving a range of sequence numbers reduces the number of updates to the transaction log. The range of sequence numbers is called a *cache*. Sequence updates on the active database are replicated only when they are followed by or used in replicated transactions.

Consider a sequence named `my.sequence` with a `MINVALUE` of 1, an `INCREMENT` of 1 and the default *`Cache`* of 20. The very first time that you reference `my.sequence.NEXTVAL`, the current value of the sequence on the active database is changed to 2, and a new current value of 21 (20+1) is replicated to the standby database. The next 19 references to `my.seq.NEXTVAL` on the active database result in no new current value being replicated, because the current value of 21 on the standby database is still ahead of the current value on the active database. On the twenty-first reference to `my.seq.NEXTVAL`, a new current value of 41 (21+20) is transmitted to the standby database because the previous current value of 21 on the standby database is now behind the value of 22 on the active database.

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 transactions on replicated tables. A side effect of this behavior is that these sequence updates are not purged from the log until followed by transactions on replicated tables. This causes `ttRepSubscriberWait` and `ttRepAdmin -wait` to fail when only these sequence updates are present at the end of the log.

4

Administering an Active Standby Pair Without Cache Groups

This chapter describes how to administer an active standby pair that does not replicate cache groups.

For information about administering active standby pairs that replicate cache groups, see Chapter 5, "Administering an Active Standby Pair with Cache Groups".

For information about managing failover and recovery automatically, see Chapter 7, "Using Oracle Clusterware to Manage Active Standby Pairs".

This chapter includes the following topics:

- Overview of master database states
- Duplicating a database
- Setting up an active standby pair with no cache groups
- Recovering from a failure of the active database
- Recovering from a failure of the standby database
- Recovering from the failure of a subscriber database
- Reversing the roles of the active and standby databases
- Detection of dual active databases

Overview of master database states

This section summarizes the possible states of a master database. These states are referenced in the tasks described in the rest of the chapter.

The master databases can be in one of the following states:

- `ACTIVE` - A database in this state is the active database. Applications can update its replicated tables.
- `STANDBY` - A database in this state is the standby database. Applications can update only nonreplicated tables in the standby database. Nonreplicated tables are tables that have been excluded from the replication scheme by using the `EXCLUDE TABLE` or `EXCLUDE CACHE GROUP` clauses of the `CREATE ACTIVE STANDBY PAIR` statement.
- `FAILED` - A database in this state is a failed master database. No updates can be replicated to it.

- IDLE - A database in this state has not yet had its role in the active standby pair assigned. It cannot be updated. Every database comes up in the IDLE state.
- RECOVERING - When a previously failed master database is synchronizing updates with the active database, it is in the RECOVERING state.

You can use the ttRepStateGet built-in procedure to discover the state of a master database.

Duplicating a database

When you set up a replication scheme or administer a recovery, a common task is duplicating a database. You can use the ttRepAdmin -duplicate utility or the ttRepDuplicateEx C function to duplicate a database.

To duplicate a database, these conditions must be fulfilled:

- The instance administrator performs the duplicate operation.
- The instance administrator user name must be the same on both instances involved in the duplication.
- You must provide the user name and password for a user with the ADMIN privilege on the source database.
- The target DSN cannot include client/server attributes.

On the source database, create a user and grant the ADMIN privilege to the user:

```
CREATE USER ttuser IDENTIFIED BY ttuser;
User created.

GRANT admin TO ttuser;
```

Assume the user name of the instance administrator is timesten. Logged in as timesten on the target host, duplicate database dsn1 on host1 to dsn2:

```
ttRepAdmin -duplicate -from dsn1 -host host1 dsn2

Enter internal UID at the remote datastore with ADMIN privileges: ttuser
Enter password of the internal Uid at the remote datastore:
```

Enter ttuser when prompted for the password of the internal user at the remote database.

If you are duplicating an active database that has cache groups, use the -keepCG option. You must also specify the cache administration user ID and password with the -cacheUid and -cachePwd options. If you do not provide the cache administration user password, ttRepAdmin prompts for a password. If the cache administration user ID is orauser and the password is orapwd, duplicate database dsn1 on host1:

```
ttRepAdmin -duplicate -from dsn1 -host host1 -keepCG "DSN=dsn2;UID=;PWD="

Enter internal UID at the remote datastore with ADMIN privileges: ttuser
Enter password of the internal Uid at the remote datastore:
```

Enter ttuser when prompted for the password. ttRepAdmin then prompts for the cache administration user and password:

```
Enter cache administrator UID: orauser
Enter cache administrator password:
```

Enter orapwd when prompted for the cache administration password.

The UID and PWD for dsn2 are specified as null values in the connection string so that the connection is made as the current OS user, which is the instance administrator. Only the instance administrator can run ttRepAdmin -duplicate. If dsn2 is configured with PWDCrypt instead of PWD, then the connection string should be "DSN=dsn2;UID=;PWDCrypt=".

When you duplicate a standby database with cache groups to a read-only subscriber, use the -nokeepCG option. In this example, dsn2 is the standby database and sub1 is the read-only subscriber:

```
ttRepAdmin -duplicate -from dsn2 -host host2 -nokeepCG "DSN=sub1;UID=;PWD="
```

The ttRepAdmin utility prompts for values for -uid and -pwd.

If you want to use a specific local or remote network interface over which the database duplication occurs, you can optionally specify either by providing an alias or the IP address of the network interface.

You can specify the local and remote network interfaces for the source and target hosts by using the -localIP and -remoteIP options of ttRepAdmin -duplicate. If you do not specify one or both network interfaces, TimesTen chooses them.

For more information about the ttRepAdmin utility, see "ttRepAdmin" in *Oracle TimesTen In-Memory Database Reference*. For more information about the ttRepDuplicateEx C function, see "ttRepDuplicateEx" in *Oracle TimesTen In-Memory Database C Developer's Guide*.

Setting up an active standby pair with no cache groups

To set up an active standby pair, complete the tasks in this section. See "Configuring an active standby pair with one subscriber" on page 2-1 for an example.

If you intend to replicate read-only cache groups or asynchronous writethrough (AWT) cache groups, see Chapter 5, "Administering an Active Standby Pair with Cache Groups".

Before you create a database, see the information in these sections:

- "Configuring the network" on page 10-1
- "Connection attributes for replicated databases" on page 10-6
- "Managing the transaction log on a replicated database" on page 10-9

1. Create a database. See "Managing TimesTen Databases" in *Oracle TimesTen In-Memory Database Operations Guide*.
2. Create the replication scheme using the CREATE ACTIVE STANDBY PAIR statement. See Chapter 3, "Defining an Active Standby Pair Replication Scheme".
3. Call ttRepStateSet('ACTIVE') on the active database.
4. Start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
5. Create a user on the active database and grant the ADMIN privilege to the user.
6. Duplicate the active database to the standby database.
7. Start the replication agent on the standby database. See "Starting and stopping the replication agents" on page 10-13.
8. Wait for the standby database to enter the STANDBY state. Use the ttRepStateGet procedure to check the state of the standby database.

9. Duplicate all of the subscribers from the standby database. See "Duplicating a master database to a subscriber" on page 10-12.
10. Set up the replication agent policy and start the replication agent on each of the subscriber databases. See "Starting and stopping the replication agents" on page 10-13.

Recovering from a failure of the active database

This section includes the following topics:

- Recovering when the standby database is ready
- Recovering when the standby database is not ready
- Failing back to the original nodes

Recovering when the standby database is ready

This section describes how to recover the active database when the standby database is available and synchronized with the active database. 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 database if it has not already been stopped.
2. On the standby database, execute `ttRepStateSet('ACTIVE')`. This changes the role of the database from `STANDBY` to `ACTIVE`.
3. On the new active database, execute `ttRepStateSave('FAILED', 'failed_database','host_name')`, where *`failed_database`* is the former active database that failed. This step is necessary for the new active database to replicate directly to the subscriber databases. During normal operation, only the standby database replicates to the subscribers.
4. Destroy the failed database.
5. Duplicate the new active database to the new standby database.
6. Set up the replication agent policy and start the replication agent on the new standby database. See "Starting and stopping the replication agents" on page 10-13.

The standby database contacts the active database. The active database stops sending updates to the subscribers. When the standby database is fully synchronized with the active database, then the standby database enters the `STANDBY` state and starts sending updates to the subscribers.

Note: You can verify that the standby database has entered the `STANDBY` state by using the `ttRepStateGet` built-in procedure.

When replication is return twosafe

Complete the following tasks:

1. On the standby database, execute `ttRepStateSet('ACTIVE')`. This changes the role of the database from `STANDBY` to `ACTIVE`.
2. On the new active database, execute `ttRepStateSave('FAILED', 'failed_database','host_name')`, where *failed_database* is the former active database that failed. This step is necessary for the new active database to replicate directly to the subscriber databases. During normal operation, only the standby database replicates to the subscribers.
3. Connect to the failed database. This triggers recovery from the local transaction logs. If database 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 4-4.
4. Verify that the replication agent for the failed database has restarted. If it has not restarted, then start the replication agent. See "Starting and stopping the replication agents" on page 10-13.

When the active database determines that it is fully synchronized with the standby database, then the standby database enters the `STANDBY` state and starts sending updates to the subscribers.

> **Note:** You can verify that the standby database has entered the `STANDBY` state by using the `ttRepStateSet` built-in procedure.

Recovering when the standby database is not ready

Consider the following scenarios:

- The standby database fails. The active database fails before the standby comes back up or before the standby has been synchronized with the active database.
- The active database fails. The standby database becomes `ACTIVE`, and the rest of the recovery process begins. (See "Recovering from a failure of the active database" on page 4-4.) The new active database fails before the new standby database is fully synchronized with it.

In both scenarios, the subscribers may have had more changes applied than the standby database.

When the active database fails and the standby database has not applied all of the changes that were last sent from the active database, there are two choices for recovery:

- Recover the *active* database from the local transaction logs.
- Recover the *standby* database from the local transaction logs.

The choice depends on which database is available and which is more up to date.

Recover the active database

1. Connect to the failed active database. This triggers recovery from the local transaction logs.
2. Verify that the replication agent for the failed active database has restarted. If it has not restarted, then start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
3. Execute `ttRepStateSet('ACTIVE')` on the newly recovered database.

4. Continue with Step 6 in "Setting up an active standby pair with no cache groups" on page 4-3.

Recover the standby database

1. Connect to the failed standby database. This triggers recovery from the local transaction logs.
2. If the replication agent for the standby database has automatically restarted, you must stop the replication agent. See "Starting and stopping the replication agents" on page 10-13.
3. Drop the replication configuration using the `DROP ACTIVE STANDBY PAIR` statement.
4. Re-create the replication configuration using the `CREATE ACTIVE STANDBY PAIR` statement.
5. Execute `ttRepStateSet('ACTIVE')` on the master database, giving it the `ACTIVE` role.
6. Set up the replication agent policy and start the replication agent on the new standby database. See "Starting and stopping the replication agents" on page 10-13.
7. Continue from Step 6 in "Setting up an active standby pair with no cache groups" on page 4-3.

Failing back to the original nodes

After a successful failover, you may wish to fail back so that the active database and the standby database are on their original nodes. See "Reversing the roles of the active and standby databases" on page 4-7 for instructions.

Recovering from a failure of the standby database

To recover from a failure of the standby database, complete the following tasks:

1. Detect the standby database failure.
2. If return twosafe service is enabled, the failure of the standby database may prevent a transaction in progress from being committed on the active database, resulting in error 8170, "Receipt or commit acknowledgement not returned in the specified timeout interval". If so, then call the `ttRepSyncSet` 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_database`*`','`*`host_name`*`')` on the active database. After this, as long as the standby database is unavailable, updates to the active database are replicated directly to the subscriber databases. Subscriber databases may also be duplicated directly from the active.
4. If the replication agent for the standby database has automatically restarted, stop the replication agent. See "Starting and stopping the replication agents" on page 10-13.
5. Recover the standby database in one of the following ways:

- Connect to the standby database. This triggers recovery from the local transaction logs.
- Duplicate the standby database from the active database.

The amount of time that the standby database has been down and the amount of transaction logs that need to be applied from the active database determine the method of recovery that you should use.

6. Set up the replication agent policy and start the replication agent on the new standby database. See "Starting and stopping the replication agents" on page 10-13.

The standby database enters the `STANDBY` state and starts sending updates to the subscribers after the active database determines that the two master databases have been synchronized and stops sending updates to the subscribers.

Note: You can verify that the standby database has entered the `STANDBY` state by using the `ttRepStateGet` procedure.

Recovering from the failure of a subscriber database

If a subscriber database 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 database.

If the standby database is down or in recovery, then duplicate the subscriber from the active database.

After the subscriber database has been recovered, then set up the replication agent policy and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.

Reversing the roles of the active and standby databases

To change the role of the active database to standby and vice versa:

1. Pause any applications that are generating updates on the current active database.
2. Execute `ttRepSubscriberWait` on the active database, with the DSN and host of the current standby database as input parameters. It must return success (`<00>`). This ensures that all updates have been transmitted to the current standby database.
3. Stop the replication agent on the current active database. See "Starting and stopping the replication agents" on page 10-13.
4. Execute `ttRepDeactivate` on the current active database. This puts the database in the `IDLE` state.
5. Execute `ttRepStateSet('ACTIVE')` on the current standby database. This database now acts as the active database in the active standby pair.
6. Set up the replication agent policy and start the replication agent on the old active database.

7. Use the `ttRepStateGet` procedure to determine when the database's state has changed from `IDLE` to `STANDBY`. The database now acts as the standby database in the active standby pair.
8. Resume any applications that were paused in Step 1.

Detection of dual active databases

Ordinarily, the designation of the active and standby databases in an active standby pair is explicitly controlled by the user. However, in some circumstances the user may not have the ability to modify both the active and standby databases when changing the role of the standby database to active.

For example, if network communication to the site of an active database is interrupted, the user may need the standby database at a different site to take over the role of the active, but cannot stop replication on the current active or change its role manually. Changing the standby database to active without first stopping replication on the active leads to a situation where both masters are in the `ACTIVE` state and accepting transactions. In such a scenario, TimesTen can automatically negotiate the active/standby role of the master databases when network communication between the databases is restored.

If, during the initial handshake between the databases, TimesTen determines that the master databases in an active standby pair replication scheme are both in the `ACTIVE` state, TimesTen performs the following operations automatically:

- The database which was set to the `ACTIVE` state most recently is left in the `ACTIVE` state and may continue to be connected to and updated by applications.
- The database which was set to the `ACTIVE` state least recently is invalidated. All applications are disconnected.
- When the invalidated database comes back up, TimesTen determines whether any transactions have occurred on the database that have not yet been replicated to the other master database. If such transactions have occurred, they are now trapped, and the database is left in the `IDLE` state. The database needs to be duplicated from the active in order to become a standby. If there are no trapped transactions, the database is safe to use as a standby database and is automatically set to the `STANDBY` state.

5

Administering an Active Standby Pair with Cache Groups

You can replicate tables within cache groups as long as they are configured within an active standby pair.

> **Note:** For information about managing failover and recovery automatically, see Chapter 7, "Using Oracle Clusterware to Manage Active Standby Pairs".

The following sections describe how to administer an active standby pair that replicates cache groups:

- Active standby pairs with cache groups
- Setting up an active standby pair with a read-only cache group
- Setting up an active standby pair with an AWT cache group
- Recovering from a failure of the active database
- Recovering from a failure of the standby database
- Recovering from the failure of a subscriber database
- Reversing the roles of the active and standby databases
- Detection of dual active databases
- Using a disaster recovery subscriber in an active standby pair

Active standby pairs with cache groups

An active standby pair that replicates a read-only cache group or an asynchronous writethrough (AWT) cache group can change the role of the cache group automatically as part of failover and recovery. This helps ensure high availability of cache instances with minimal data loss. See "Replicating an AWT cache group" on page 1-12 and "Replicating a read-only cache group" on page 1-13.

> **Note:** TimesTen does not support replication of a user managed cache group if it is defined with either the `AUTOREFRESH` or `PROPAGATE` cache table attributes.

You can also create a special disaster recovery read-only subscriber when you set up active standby replication of an AWT cache group. This special subscriber, located at a remote disaster recovery site, can propagate updates to a second Oracle database, also located at the disaster recovery site. See "Using a disaster recovery subscriber in an active standby pair" on page 5-9.

You cannot use an active standby pair to replicate synchronous writethrough (SWT) cache groups. If you are using an active standby pair to replicated a database with SWT cache groups, you must either drop or exclude the SWT cache groups.

Setting up an active standby pair with a read-only cache group

This section describes how to set up an active standby pair that replicates cache tables in a read-only cache group. The active standby pair used as an example in this section is not a cache grid member.

Before you create a database, see the information in these sections:

- "Configuring the network" on page 10-1
- "Connection attributes for replicated databases" on page 10-6
- "Managing the transaction log on a replicated database" on page 10-9

To set up an active standby pair that replicates a local read-only cache group, complete the following tasks:

1. Create a cache administration user in the Oracle database. See "Create users in the Oracle database" in *Oracle In-Memory Database Cache User's Guide*.
2. Create a database. See "Create a DSN for the TimesTen database" in *Oracle In-Memory Database Cache User's Guide*.
3. Set the cache administration user ID and password by calling the `ttCacheUidPwdSet` built-in procedure. See "Set the cache administration user name and password in the TimesTen database" in *Oracle In-Memory Database Cache User's Guide*. For example:

```
Command> call ttCacheUidPwdSet('orauser','orapwd');
```

4. Start the cache agent on the database. Use the `ttCacheStart` built-in procedure or the `ttAdmin -cachestart` utility.

```
Command> call ttCacheStart;
```

5. Use the `CREATE CACHE GROUP` statement to create the read-only cache group. For example:

```
Command> CREATE READONLY CACHE GROUP readcache
       > AUTOREFRESH INTERVAL 5 SECONDS
       > FROM oratt.readtab
       > (keyval NUMBER NOT NULL PRIMARY KEY, str VARCHAR2(32));
```

6. Ensure that the autorefresh state is set to `PAUSED`. The autorefresh state is `PAUSED` by default after cache group creation. You can verify the autorefresh state by executing the `ttIsql cachegroups` command:

```
Command> cachegroups;
```

7. Create the replication scheme using the `CREATE ACTIVE STANDBY PAIR` statement.

For example, suppose master1 and master2 are defined as the master databases. sub1 and sub2 are defined as the subscriber databases. The databases reside on node1, node2, node3, and node4. The return service is RETURN RECEIPT. The replication scheme can be specified as follows:

```
Command> 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;
```

8. Set the replication state to ACTIVE by calling the ttRepStateSet built-in procedure on the active database (master1). For example:

```
Command> call ttRepStateSet('ACTIVE');
```

9. Set up the replication agent policy for master1 and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.

10. Load the cache group by using the LOAD CACHE GROUP statement. This starts the autorefresh process. For example:

```
Command> LOAD CACHE GROUP readcache COMMIT EVERY 256 ROWS;
```

11. As the instance administrator, duplicate the active database (master1) to the standby database (master2). Use the ttRepAdmin -duplicate utility with the -keepCG option to preserve the cache group. Alternatively, you can use the ttRepDuplicateEx C function to duplicate the database. See "Duplicating a database" on page 4-2. ttRepAdmin prompts for the values of -uid, -pwd, -cacheuid and -cachepwd.

```
ttRepAdmin -duplicate -from master1 -host node1 -keepCG "DSN=master2;UID=;PWD="
```

12. Set up the replication agent policy on master2 and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.

13. The standby database enters the STANDBY state automatically. Wait for master2 to enter the STANDBY state. Call the ttRepStateGet built-in procedure to check the state of master2. For example:

```
Command> call ttRepStateGet;
```

14. Start the cache agent for master2 using the ttCacheStart built-in procedure or the ttAdmin -cacheStart utility. For example:

```
Command> call ttCacheStart;
```

15. As the instance administrator, duplicate the subscribers (sub1 and sub2) from the standby database (master2). Use the -noKeepCG command line option with ttRepAdmin -duplicate to convert the cache tables to normal TimesTen tables on the subscribers. ttRepAdmin prompts for the values of -uid and -pwd. See "Duplicating a database" on page 4-2. For example:

```
ttRepAdmin -duplicate -from master2 -host node2 -nokeepCG "DSN=sub1;UID=;PWD="
```

16. Set up the replication agent policy on the subscribers and start the replication agent on each of the subscriber databases. See "Starting and stopping the replication agents" on page 10-13.

Setting up an active standby pair with an AWT cache group

For detailed instructions for setting up an active standby pair with a global AWT cache group, see "Replicating cache tables" in *Oracle In-Memory Database Cache User's Guide*. The active standby pair in that section is a cache grid member.

Recovering from a failure of the active database

This section includes the following topics:

- Recovering when the standby database is ready
- Recovering when the standby database is not ready
- Failing back to the original nodes

Recovering when the standby database is ready

This section describes how to recover the active database when the standby database is available and synchronized with the active database. 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 database if it has not already been stopped.
2. On the standby database, execute `ttRepStateSet('ACTIVE')`. This changes the role of the database from `STANDBY` to `ACTIVE`. If you are replicating a read-only cache group, this action automatically causes the autorefresh state to change from `PAUSED` to `ON` for this database.
3. On the new active database, execute `ttRepStateSave('FAILED', 'failed_database','host_name')`, where *`failed_database`* is the former active database that failed. This step is necessary for the new active database to replicate directly to the subscriber databases. During normal operation, only the standby database replicates to the subscribers.
4. Stop the cache agent on the failed database if it is not already stopped.
5. Destroy the failed database.
6. Duplicate the new active database to the new standby database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to duplicate a database. Use the `-keepCG -recoveringNode` command line options with `ttRepAdmin` to preserve the cache group. See "Duplicating a database" on page 4-2.
7. Set up the replication agent policy on the new standby database and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
8. Start the cache agent on the new standby database.

The standby database contacts the active database. The active database stops sending updates to the subscribers. When the standby database is fully synchronized with the active database, then the standby database enters the `STANDBY` state and starts sending updates to the subscribers.The new standby database takes over processing of the cache group automatically when it enters the `STANDBY` state. If you are replicating

an AWT cache group, the new standby database takes over processing of the cache group automatically when it enters the STANDBY state.

Note: You can verify that the standby database has entered the STANDBY state by using the ttRepStateGet built-in procedure.

When replication is return twosafe

Complete the following tasks:

1. On the standby database, execute ttRepStateSet('ACTIVE'). This changes the role of the database from STANDBY to ACTIVE. If you are replicating a read-only cache group, this action automatically causes the autorefresh state to change from PAUSED to ON for this database.
2. On the new active database, execute ttRepStateSave('FAILED', '*failed_database*','*host_name*'), where *failed_database* is the former active database that failed. This step is necessary for the new active database to replicate directly to the subscriber databases. During normal operation, only the standby database replicates to the subscribers.
3. Connect to the failed database. This triggers recovery from the local transaction logs. If database 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 5-4. If you are replicating a read-only cache group, the autorefresh state is automatically set to PAUSED.
4. Verify that the replication agent for the failed database has restarted. If it has not restarted, then start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
5. Verify that the cache agent for the failed database has restarted. If it has not restarted, then start the cache agent.

When the active database determines that it is fully synchronized with the standby database, then the standby database enters the STANDBY state and starts sending updates to the subscribers. The new standby database takes over processing of the cache group automatically when it enters the STANDBY state. If you are replicating an AWT cache group, the new standby database takes over processing of the cache group automatically when it enters the STANDBY state.

Note: You can verify that the standby database has entered the STANDBY state by using the ttRepStateSet built-in procedure.

Recovering when the standby database is not ready

Consider the following scenarios:

- The standby database fails. The active database fails before the standby comes back up or before the standby has been synchronized with the active database.
- The active database fails. The standby database becomes ACTIVE, and the rest of the recovery process begins. (See "Recovering from a failure of the active database" on page 5-4.) The new active database fails before the new standby database is fully synchronized with it.

In both scenarios, the subscribers may have had more changes applied than the standby database.

When the active database fails and the standby database has not applied all of the changes that were last sent from the active database, there are two choices for recovery:

- Recover the *active* master database from the local transaction logs.
- Recover the *standby* master database from the local transaction logs.

The choice depends on which database is available and which is more up to date.

Recover the active database

1. Connect to the failed active database. This triggers recovery from the local transaction logs. If you are replicating a read-only cache group, the autorefresh state is automatically set to `PAUSED`.
2. Verify that the replication agent for the failed active database has restarted. If it has not restarted, then start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
3. Execute `ttRepStateSet('ACTIVE')` on the newly recovered database. If you are replicating a read-only cache group, this action automatically causes the autorefresh state to change from `PAUSED` to `ON` for this database.
4. Verify that the cache agent for the failed database has restarted. If it has not restarted, then start the cache agent.
5. Duplicate the active database to the standby database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to duplicate a database. Use the `-keepCG` command line option with `ttRepAdmin` to preserve the cache group. "Duplicating a database" on page 4-2.
6. Set up the replication agent policy on the standby database and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
7. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` procedure to check the state.
8. Start the cache agent for on the standby database using the `ttCacheStart` procedure or the `ttAdmin -cacheStart` utility.
9. Duplicate all of the subscribers from the standby database. See "Duplicating a master database to a subscriber" on page 10-12. Use the `-noKeepCG` command line option with `ttRepAdmin` in order to convert the cache group to regular TimesTen tables on the subscribers.
10. Set up the replication agent policy on the subscribers and start the agent on each of the subscriber databases. See "Starting and stopping the replication agents" on page 10-13.

Recover the standby database

1. Connect to the failed standby database. This triggers recovery from the local transaction logs. If you are replicating a read-only cache group, the autorefresh state is automatically set to `PAUSED`.
2. If the replication agent for the standby database has automatically restarted, you must stop the replication agent. See "Starting and stopping the replication agents" on page 10-13.
3. If the cache agent has automatically restarted, stop the cache agent.
4. Drop the replication configuration using the `DROP ACTIVE STANDBY PAIR` statement.

5. Drop and re-create all 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. Execute `ttRepStateSet('ACTIVE')` on the master database, giving it the `ACTIVE` role. If you are replicating a read-only cache group, this action automatically causes the autorefresh state to change from `PAUSED` to `ON` for this database.
8. Set up the replication agent policy and start the replication agent on the new active database. See "Starting and stopping the replication agents" on page 10-13.
9. Start the cache agent on the new active database.
10. Duplicate the active database to the standby database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to duplicate a database. Use the `-keepCG` command line option with `ttRepAdmin` to preserve the cache group. "Duplicating a database" on page 4-2.
11. Set up the replication agent policy on the standby database and start the replication agent on the new standby database. See "Starting and stopping the replication agents" on page 10-13.
12. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` procedure to check the state.
13. Start the cache agent for the standby database using the `ttCacheStart` procedure or the `ttAdmin -cacheStart` utility.
14. Duplicate all of the subscribers from the standby database. See "Duplicating a master database to a subscriber" on page 10-12. Use the `-noKeepCG` command line option with `ttRepAdmin` in order to convert the cache group to regular TimesTen tables on the subscribers.
15. Set up the replication agent policy on the subscribers and start the agent on each of the subscriber databases. See "Starting and stopping the replication agents" on page 10-13.

Failing back to the original nodes

After a successful failover, you may wish to fail back so that the active database and the standby database are on their original nodes. See "Reversing the roles of the active and standby databases" on page 5-9 for instructions.

Recovering from a failure of the standby database

To recover from a failure of the standby database, complete the following tasks:

1. Detect the standby database failure.
2. If return twosafe service is enabled, the failure of the standby database may prevent a transaction in progress from being committed on the active database, resulting in error 8170, "Receipt or commit acknowledgement not returned in the specified timeout interval". If so, then call the `ttRepSyncSet` 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_database','host_name')` on the active database. After this, as long as the standby database is unavailable, updates to the active database are replicated directly to the subscriber databases. Subscriber databases may also be duplicated directly from the active.
4. If the replication agent for the standby database has automatically restarted, stop the replication agent. See "Starting and stopping the replication agents" on page 10-13.
5. If the cache agent has automatically restarted, stop the cache agent.
6. Recover the standby database in one of the following ways:
   - Connect to the standby database. This triggers recovery from the local transaction logs.
   - Duplicate the standby database from the active database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to duplicate a database. Use the `-keepCG -recoveringNode` command line options with `ttRepAdmin` to preserve the cache group.See "Duplicating a database" on page 4-2.

   The amount of time that the standby database has been down and the amount of transaction logs that need to be applied from the active database determine the method of recovery that you should use.
7. Set up the replication agent policy and start the replication agent on the standby database. See "Starting and stopping the replication agents" on page 10-13.
8. Start the cache agent.

The standby database enters the `STANDBY` state and starts sending updates to the subscribers after the active database determines that the two master databases have been synchronized and stops sending updates to the subscribers.

Note: You can verify that the standby database has entered the `STANDBY` state by using the `ttRepStateGet` procedure.

Recovering from the failure of a subscriber database

If a subscriber database 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 database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to duplicate a database. 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 database is down or in recovery, then duplicate the subscriber from the active database.

After the subscriber database has been recovered, then set up the replication agent policy and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.

Reversing the roles of the active and standby databases

To change the role of the active database to standby and vice versa:

1. Pause any applications that are generating updates on the current active database.
2. Execute `ttRepSubscriberWait` on the active database, with the DSN and host of the current standby database as input parameters. It must return success (`<00>`). This ensures that all updates have been transmitted to the current standby database.
3. Stop the replication agent on the current active database. See "Starting and stopping the replication agents" on page 10-13.
4. If global cache groups are not present, stop the cache agent on the current active database. When global cache groups are present, set the autorefresh state to `PAUSED`.
5. Execute `ttRepDeactivate` on the current active database. This puts the database in the `IDLE` state. If you are replicating a read-only cache group, this action automatically causes the autorefresh state to change from `ON` to `PAUSED` for this database.
6. Execute `ttRepStateSet('ACTIVE')` on the current standby database. This database now acts as the active database in the active standby pair. If you are replicating a read-only cache group, this automatically causes the autorefresh state to change from `PAUSED` to `ON` for this database.
7. Start the replication agent on the former master database.
8. Configure the replication agent policy as needed and start the replication agent on the former active database. Use the `ttRepStateGet` procedure to determine when the database's state has changed from `IDLE` to `STANDBY`. The database now acts as the standby database in the active standby pair.
9. Start the cache agent on the former active database if it is not already running.
10. Resume any applications that were paused in Step 1.

Detection of dual active databases

See "Detection of dual active databases" on page 4-8. There is no difference for active standby pairs that replicate cache groups.

Using a disaster recovery subscriber in an active standby pair

TimesTen active standby pair replication provides high availability by allowing for fast switching between databases within a data center. This includes the ability to automatically change which database propagates changes to an Oracle database using AWT cache groups. However, for additional high availability across data centers, you may require the ability to recover from a failure of an entire site, which can include a failure of both TimesTen master databases in the active standby pair as well as the Oracle database used for the cache groups.

You can recover from a complete site failure by creating a special disaster recovery read-only subscriber as part of the active standby pair replication scheme. The standby database sends updates to cache group tables on the read-only subscriber. This special subscriber is located at a remote disaster recovery site and can propagate updates to a second Oracle database, also located at the disaster recovery site. The disaster recovery subscriber can take over as the active in a new active standby pair at the disaster

recovery site if the primary site suffers a complete failure. Any applications may then connect to the disaster recovery site and continue operating, with minimal interruption of service.

Requirements for using a disaster recovery subscriber with an active standby pair

To use a disaster recovery subscriber, you must:

- Use an active standby pair configuration with AWT cache groups at the primary site. The active standby pair can also include read-only cache groups in the replication scheme. The read-only cache groups are converted to regular tables on the disaster recovery subscriber. The AWT cache group tables remain AWT cache group tables on the disaster recovery subscriber.
- Have a continuous WAN connection from the primary site to the disaster recovery site. This connection should have at least enough bandwidth to guarantee that the normal volume of transactions can be replicated to the disaster recovery subscriber at a reasonable pace.
- Configure an Oracle database at the disaster recovery site to include tables with the same schema as the database at the primary site. Note that this database is intended only for capturing the replicated updates from the primary site, and if any data exists in tables written to by the cache groups when the disaster recovery subscriber is created, that data is deleted.
- Have the same cache group administrator user ID and password at both the primary and the disaster recovery site.

Though it is not absolutely required, you should have a second TimesTen database configured at the disaster recovery site. This database can take on the role of a standby database, in the event that the disaster recovery subscriber is promoted to an active database after the primary site fails.

Rolling out a disaster recovery subscriber

To create a disaster recovery subscriber, follow these steps:

1. Create an active standby pair with AWT cache groups at the primary site. The active standby pair can also include read-only cache groups. The read-only cache groups are converted to regular tables when the disaster recovery subscriber is rolled out.
2. Create the disaster recovery subscriber at the disaster recovery site using the `ttRepAdmin` utility with the `-duplicate` and `-initCacheDR` options. You must also specify the cache group administrator and password for the Oracle database at the disaster recovery site using the `-cacheUid` and `-cachePwd` options.

   If your database includes multiple cache groups, you may improve the efficiency of the duplicate operation by using the `-nThreads` option to specify the number of threads that are spawned to flush the cache groups in parallel. Each thread flushes an entire cache group to Oracle and then moves on to the next cache group, if any remain to be flushed. If a value is not specified for `-nThreads`, only one flushing thread is spawned.

   For example, duplicate the standby database `mast2`, on the system with the host name `primary` and the cache user ID `system` and password `manager`, to the disaster recovery subscriber `drsub`, and using two cache group flushing threads. `ttRepAdmin` prompts for the values of `-uid`, `-pwd`, `-cacheUid` and `-cachePwd`.

```
ttRepAdmin -duplicate -from mast2 -host primary -initCacheDR -nThreads 2
"DSN=drsub;UID=;PWD=;"
```

If you use the `ttRepDuplicateEx` function in C, you must set the `TT_REPDUP_INITCACHEDR` flag in `ttRepDuplicateExArg.flags` and may optionally specify a value for `ttRepDuplicateExArg.nThreads4InitDR`:

```
int                 rc;
ttUtilHandle        utilHandle;
ttRepDuplicateExArg arg;
memset( &arg, 0, sizeof( arg ) );
arg.size = sizeof( ttRepDuplicateExArg );
arg.flags = TT_REPDUP_INITCACHEDR;
arg.nThreads4InitDR = 2;
arg.uid="ttuser"
arg.pwd="ttuser"
arg.cacheuid = "system";
arg.cachepwd = "manager";
arg.localHost = "disaster";
rc = ttRepDuplicateEx( utilHandle, "DSN=drsub",
                       "mast2", "primary", &arg );
```

After the subscriber is duplicated, TimesTen automatically configures the replication scheme that propagates updates from the AWT cache groups to the Oracle database, truncates the tables in the Oracle database that correspond to the cache groups in TimesTen, and then flushes all of the data in the cache groups to the Oracle database.

3. If you wish to set the failure threshold for the disaster recovery subscriber, call the `ttCacheAWTThresholdSet` built-in procedure and specify the number of transaction log files that can accumulate before the disaster recovery subscriber is considered either dead or too far behind to catch up.

   If one or both master databases had a failure threshold configured before the disaster recovery subscriber was created, then the disaster recovery subscriber inherits the failure threshold value when it is created with the `ttRepAdmin -duplicate -initCacheDR` command. If the master databases have different failure thresholds, then the higher value is used for the disaster recovery subscriber.

   For more information about the failure threshold, see "Setting the log failure threshold" on page 3-11.

4. Start the replication agent for the disaster recovery subscriber using the `ttRepStart` procedure or the `ttAdmin` utility with the `-repstart` option. For example:

   ```
   ttAdmin -repstart drsub
   ```

   Updates are now replicated from the standby database to the disaster recovery subscriber, which then propagates the updates to the Oracle database at the disaster recovery site.

Switching over to the disaster recovery site

When the primary site has failed, you can switch over to the disaster recovery site in one of two ways. If your goal is to minimize risk of data loss at the disaster recovery site, you may roll out a new active standby pair using the disaster recovery subscriber as the active database. If the goal is to absolutely minimize the downtime of your applications, at the risk of data loss if the disaster recovery database later fails, you

may instead choose to drop the replication scheme from the disaster recovery subscriber and use it as a single non-replicating database. You may deploy an active standby pair at the disaster recovery site later.

Creating a new active standby pair after switching to the disaster recovery site

1. Any read-only applications may be redirected to the disaster recovery subscriber immediately. Redirecting applications that make updates to the database must wait until Step 7.
2. Ensure that all of the recent updates to the cache groups have been propagated to the Oracle database using the `ttRepSubscriberWait` procedure or the `ttRepAdmin` command with the `-wait` option.

   ```
   ttRepSubscriberWait( null, null, '_ORACLE', null, 600 );
   ```

   It must return success (<00>). If `ttRepSubscriberWait` returns `0x01`, indicating a timeout, investigate to determine why the cache groups are not finished propagating before continuing to Step 3.
3. Stop the replication agent on the disaster recovery subscriber using the `ttRepStop` procedure or the `ttAdmin` command with the `-repstop` option. For example, to stop the replication agent for the subscriber `drsub`, use:

   ```
   call ttRepStop;
   ```

4. Drop the active standby pair replication scheme on the subscriber using the `DROP ACTIVE STANDBY PAIR` statement. For example:

   ```
   DROP ACTIVE STANDBY PAIR;
   ```

5. If there are tables on the disaster recovery subscriber that were converted from read-only cache group tables on the active database, drop the tables on the disaster recovery subscriber.
6. Create the read-only cache groups on the disaster recovery subscriber. Ensure that the autorefresh state is set to `PAUSED`.
7. Create a new active standby pair replication scheme using the `CREATE ACTIVE STANDBY PAIR` statement, specifying the disaster recovery subscriber as the active database. For example, to create a new active standby pair with the former subscriber `drsub` as the active and the new database `drstandby` as the standby, and using the return twosafe return service, use:

   ```
   CREATE ACTIVE STANDBY PAIR drsub, drstandby RETURN TWOSAFE;
   ```

8. Set the new active standby database to the `ACTIVE` state using the `ttRepStateSet` procedure. For example, on the database `drsub` in this example, execute:

   ```
   call ttRepStateSet( 'ACTIVE' );
   ```

9. Any applications which must write to the TimesTen database may now be redirected to the new active database.
10. If you are replicating a read-only cache group, load the cache group using the `LOAD CACHE GROUP` statement 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.
11. Duplicate the active database to the standby database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to

duplicate a database. Use the `-keepCG` command line option with `ttRepAdmin` to preserve the cache group. See "Duplicating a database" on page 4-2.

12. Set up the replication agent policy on the standby database and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
13. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` procedure to check the state.
14. Start the cache agent for the standby database using the `ttCacheStart` procedure or the `ttAdmin -cacheStart` utility.
15. Duplicate all of the subscribers from the standby database. See "Duplicating a master database to a subscriber" on page 10-12. Use the `-noKeepCG` command line option with `ttRepAdmin` in order to convert the cache group to regular TimesTen tables on the subscribers.
16. Set up the replication agent policy on the subscribers and start the agent on each of the subscriber databases. See "Starting and stopping the replication agents" on page 10-13.

Switching over to a single database

1. Any read-only applications may be redirected to the disaster recovery subscriber immediately. Redirecting applications that make updates to the database must wait until Step 5.
2. Stop the replication agent on the disaster recovery subscriber using the `ttRepStop` procedure or the `ttAdmin` command with the `-repstop` option. For example, to stop the replication agent for the subscriber drsub, use:

   ```
   call ttRepStop;
   ```

3. Drop the active standby pair replication scheme on the subscriber using the `DROP ACTIVE STANDBY PAIR` statement. For example:

   ```
   DROP ACTIVE STANDBY PAIR;
   ```

4. If there are tables on the disaster recovery subscriber that were converted from read-only cache group tables on the active database, drop the tables on the disaster recovery subscriber.
5. Create the read-only cache groups on the disaster recovery subscriber.
6. Although there is no longer an active standby pair configured, AWT cache groups require the replication agent to be started. Start the replication agent on the database using the `ttRepStart` procedure or the `ttAdmin` command with the `-repstart` option. For example, to start the replication agent for the database `drsub`, use:

   ```
   call ttRepStart;
   ```

7. Any applications which must write to a TimesTen database may now be redirected to the this database.

Note: You may choose to roll out an active standby pair at the disaster recovery site at a later time. You may do this by following the steps in "Creating a new active standby pair after switching to the disaster recovery site" on page 5-12, starting at Step 2 and skipping Step 4.

Returning to the original configuration at the primary site

When the primary site is usable again, you may wish to move the working active standby pair from the disaster recovery site back to the primary site. You can do this with a minimal interruption of service by reversing the process that was used to create and switch over to the original disaster recovery site. Follow these steps:

1. Destroy original active database at the primary site, if necessary, using the `ttDestroy` utility. For example, to destroy a database called `mast1`, use:

   ```
   ttDestroy mast1
   ```

2. Create a disaster recovery subscriber at the primary site, following the steps detailed in "Rolling out a disaster recovery subscriber" on page 5-10. Use the original active database for the new disaster recovery subscriber.
3. Switch over to the new disaster recovery subscriber at primary site, as detailed in "Switching over to the disaster recovery site" on page 5-11. Roll out the standby database as well.
4. Roll out a new disaster recovery subscriber at the disaster recovery site, as detailed in "Rolling out a disaster recovery subscriber" on page 5-10.

6

Altering an Active Standby Pair

This chapter includes the following sections:

Making DDL changes in an active standby pair

You can perform the following tasks in an active standby pair without stopping the replication agent:

- Create, alter, or drop a user. These statements are replicated.
- Grant or revoke privileges from a user. These statements are replicated.
- Create or drop a view, a materialized view, a PL/SQL function, PL/SQL procedure, PL/SQL package, or PL/SQL package body. These objects are not replicated. See "Creating a new PL/SQL object in an existing active standby pair" on page 6-2 for more information.
- Add a column to a replicated table or drop a column from a replicated table. The change is replicated to the table in the standby database.
- Create or drop a table, including global temporary tables. The `CREATE TABLE` and `DROP TABLE` statements can be replicated to the standby database. The new table can also be included in the active standby pair.
- Create or drop a synonym. The `CREATE SYNONYM` and `DROP SYNONYM` statements can be replicated to the standby database.
- Create or drop an index. The `CREATE INDEX` and `DROP INDEX` statements can be replicated to the standby database.

Use the `DDLReplicationLevel` and `DDLReplicationAction` connection attributes to control what happens when you want to perform these tasks.

`DDLReplicationLevel` can be set as follows:

- `DDLReplicationLevel=1`. `CREATE` or `DROP` statements for tables, indexes, or synonyms are not replicated to the standby database. However, you can add to or drop columns from a replicated table, and those actions will be replicated to the standby database.
- `DDLReplicationLevel=2` is the default, which enables replication of creating and dropping of tables, indexes, and synonyms.

You can set the `DDLReplicationLevel` attribute by using the `ALTER SESSION` statement:

```
ALTER SESSION SET ddl_replication_level=1;
```

If you want to include a table in the active standby pair when the table is created, set the DDLReplicationAction connection attribute to 'INCLUDE'. If you do not want to include a table in the active standby pair when the table is created, set DDLReplicationAction='EXCLUDE'. The default is 'INCLUDE'.

You can set the DDLReplicationAction attribute by using the ALTER SESSION statement:

```
ALTER SESSION SET ddl_replication_action='EXCLUDE';
```

To add an existing table to an active standby pair, set DDLReplicationLevel=2 and use the ALTER ACTIVE STANDBY PAIR INCLUDE TABLE statement. The table must be empty.

When DDLCommitBehavior=0 (the default), DDL operations are automatically committed. When RETURN TWOSAFE has been specified, errors and timeouts may occur as described in "RETURN TWOSAFE" on page 3-5. If a RETURN TWOSAFE timeout occurs, the DDL transaction is committed locally regardless of the LOCAL COMMIT ACTION that has been specified.

Creating a new PL/SQL object in an existing active standby pair

To add a new PL/SQL procedure, package, package body or function to an existing active standby pair, complete these tasks:

1. Create the PL/SQL object on the active database. The CREATE statement is not replicated to the standby database.
2. Create the PL/SQL object on the standby database.
3. Grant privileges to the new PL/SQL object on the active database. The GRANT statement is replicated to the standby database.

Restrictions on making DDL changes in an active standby pair

- CREATE TABLE AS SELECT is not replicated.
- The CREATE INDEX statement is replicated only when the index is created on an empty table.
- These statements cannot be executed on the standby database when DDLReplicationLevel=2:
  - CREATE USER, ALTER USER, DROP USER
  - GRANT, REVOKE
  - CREATE SYNONYM, DROP SYNONYM

Examples: Making DDL changes in an active standby pair

Example 6–1 Create a table and include it in the active standby pair

On the active database, set DDLReplicationLevel to 2 and DDLReplicationAction to 'INCLUDE'.

```
Command > ALTER SESSION SET ddl_replication_level=2;
Session altered.
Command > ALTER SESSION SET ddl_replication_action='INCLUDE';
Session altered.
```

Create a table. The table must have a primary key or index.

```
Command > CREATE TABLE tabinclude (col1 NUMBER NOT NULL PRIMARY KEY);
Table created.
```

Insert a row into `tabinclude`.

```
Command > INSERT INTO tabinclude VALUES (55);
1 row inserted.
```

On the standby database, verify that the `INSERT` statement has been replicated. This indicates that the `tabinclude` table has been included in the active standby pair.

```
Command > SELECT * FROM tabinclude;
< 55 >
1 row found.
```

Alternatively, use the `ttIsql repschemes` command to see what tables are included in the active standby pair.

Example 6–2 Create a table and add it to the active standby pair later

On the active database, set `DDLReplicationLevel` to `2` and `DDLReplicationAction` to `'EXCLUDE'`.

```
Command> ALTER SESSION SET ddl_replication_level=2;
Session altered.
Command> ALTER SESSION SET ddl_replication_action='exclude';
Session altered.
```

Create a table that does not have a primary key or index. Try to include it in the active standby pair.

```
Command> CREATE TABLE newtab (a NUMBER NOT NULL);
Command> ALTER ACTIVE STANDBY PAIR INCLUDE TABLE newtab;
 8000: No primary or unique index on non-nullable column found for replicated
 table TERRY.NEWTAB
The command failed.
```

Create an index on the table. Include the table in the active standby pair.

```
Command> CREATE UNIQUE INDEX ixnewtab ON newtab(a);
Command> ALTER ACTIVE STANDBY PAIR INCLUDE TABLE newtab;
```

Insert a row into the table.

```
Command> INSERT INTO newtab VALUES (5);
1 row inserted.
```

On the standby database, verify that the row was inserted.

```
Command> SELECT * FROM newtab;
< 5 >
1 row found.
```

This example illustrates that a table does not need a primary key to be part of an active standby pair.

Example 6–3 CREATE INDEX is replicated

On the active database, set `DDLReplicationLevel=2` and `DDLReplicationAction='INCLUDE'`.

```
Command> ALTER SESSION SET ddl_replication_level=2;
Session altered.
Command> ALTER SESSION SET ddl_replication_action='include';
Session altered.
```

Create a table with a primary key. The table is automatically included in the active standby pair.

```
Command> CREATE TABLE tab2 (a NUMBER NOT NULL, b NUMBER NOT NULL,
       > PRIMARY KEY (a));
```

Create an index on the table.

```
Command> CREATE UNIQUE INDEX ixtab2 ON tab2 (b);
```

On the standby database, verify that the `CREATE INDEX` statement has been replicated.

```
Command> indexes;

Indexes on table TERRY.TAB2:
  IXTAB2: unique T-tree index on columns:
    B
  TAB2: unique T-tree index on columns:
    A
  2 indexes found.

Indexes on table TERRY.NEWTAB:
  NEWTAB: unique T-tree index on columns:
    A
  1 index found.

Indexes on table TERRY.TABINCLUDE:
  TABINCLUDE: unique T-tree index on columns:
    A
  1 index found.
4 indexes found on 3 tables.
```

Example 6–4 CREATE SYNONYM is replicated

On the active database, set `DDLReplicationLevel` to `2` and `DDLReplicationAction` to `'INCLUDE'`.

```
Command > ALTER SESSION SET ddl_replication_level=2;
Session altered.
Command > ALTER SESSION SET ddl_replication_action='INCLUDE';
Session altered.
```

Create a synonym for `tabinclude`.

```
Command> CREATE SYNONYM syntabinclude FOR tabinclude;
Synonym created.
```

On the standby database, use the `ttIsql synonyms` command to verify that the `CREATE SYNONYM` statement has been replicated.

```
Command> synonyms;
TERRY.SYNTABINCLUDE
1 synonym found.
```

Making other changes to an active standby pair

You must stop the replication agent to make these changes to an active standby pair:

- Include or exclude a sequence
- Include or exclude a cache group
- Add or drop a subscriber
- Change values in the `STORE` clause
- Change network operations (`ADD ROUTE` or `DROP ROUTE` clause)

To alter an active standby pair according to the preceding list, complete the following tasks:

1. Stop the replication agent on the active database. See "Starting and stopping the replication agents" on page 10-13.
2. If the active standby pair includes cache groups, stop the cache agent on the active database.
3. Use the `ALTER ACTIVE STANDBY PAIR` statement to make changes to the replication scheme. See "Examples: Altering an active standby pair" on page 6-6.
4. Start the replication agent on the active database. See "Starting and stopping the replication agents" on page 10-13.
5. If the active standby pair includes cache groups, start the cache agent on the active database.
6. Destroy the standby database and the subscribers.
7. Duplicate the active database to the standby database. You can use either the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function to duplicate a database. If the active standby pair includes cache groups, use the `-keepCG` command line option with `ttRepAdmin` to preserve the cache group. See "Duplicating a database" on page 4-2.
8. Set up the replication agent policy on the standby database and start the replication agent. See "Starting and stopping the replication agents" on page 10-13.
9. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` procedure to check the state.
10. If the active standby pair includes cache groups, start the cache agent for the standby database using the `ttCacheStart` procedure or the `ttAdmin -cacheStart` utility.
11. Duplicate all of the subscribers from the standby database. See "Duplicating a master database to a subscriber" on page 10-12. If the active standby pair includes cache groups, use the `-noKeepCG` command line option with `ttRepAdmin` in order to convert the cache group to regular TimesTen tables on the subscribers. See "Duplicating a database" on page 4-2.
12. Set up the replication agent policy on the subscribers and start the agent on each of the subscriber databases. See "Starting and stopping the replication agents" on page 10-13.

Examples: Altering an active standby pair

Example 6–5 Adding a subscriber to an active standby pair

Add a subscriber database to the active standby pair.

```
ALTER ACTIVE STANDBY PAIR
  ADD SUBSCRIBER sub1;
```

Example 6–6 Dropping subscribers from an active standby pair

Drop subscriber databases from the active standby pair.

```
ALTER ACTIVE STANDBY PAIR
  DROP SUBSCRIBER sub1
  DROP SUBSCRIBER sub2;
```

Example 6–7 Changing the PORT and TIMEOUT settings for subscribers

Alter the `PORT` and `TIMEOUT` settings for subscribers `sub1` and `sub2`.

```
ALTER ACTIVE STANDBY PAIR
  ALTER STORE sub1 SET PORT 23000 TIMEOUT 180
  ALTER STORE sub2 SET PORT 23000 TIMEOUT 180;
```

Example 6–8 Adding a cache group to an active standby pair

Add a cache group to the active standby pair.

```
ALTER ACTIVE STANDBY PAIR
  INCLUDE CACHE GROUP cg0;
```

7

Using Oracle Clusterware to Manage Active Standby Pairs

Oracle Clusterware monitors and controls applications to provide high availability. This chapter describes how to use Oracle Clusterware to manage availability for a TimesTen active standby pair.

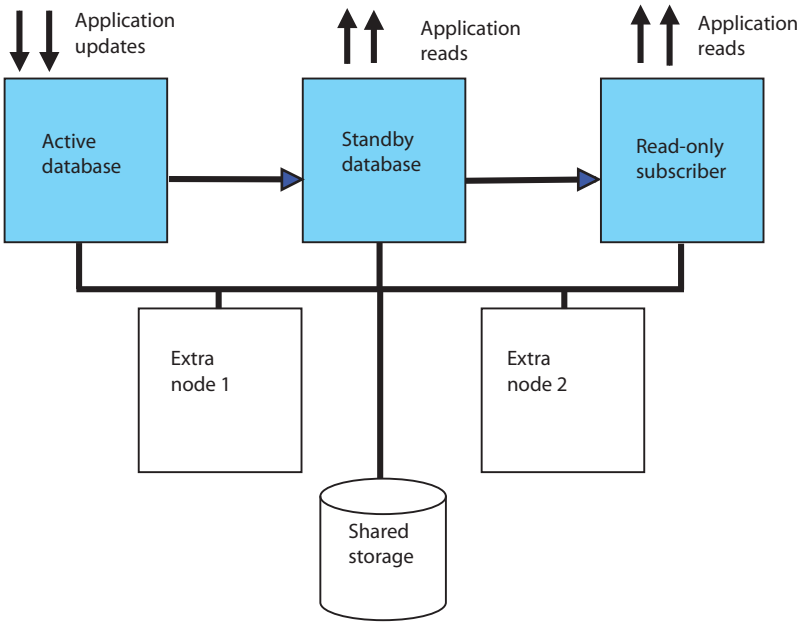
> **Note:** For more information about Oracle Clusterware, see *Oracle Clusterware Administration and Deployment Guide* in the Oracle Database documentation.

This chapter includes the following topics:

Overview

Figure 7–1 shows an active standby pair with one read-only subscriber in the same local network. The active database, the standby database and the read-only subscriber are on different nodes. There are two nodes that are not part of the active standby pair that are also running TimesTen. An application updates the active database. An application reads from the standby and the subscriber. All of the nodes are connected to shared storage.

Figure 7–1 Active standby pair with one subscriber



You can use Oracle Clusterware to start, monitor and automatically fail over TimesTen databases and applications in response to node failures and other events. See "Planned maintenance" on page 7-25 and "Recovering from failures" on page 7-17 for details.

Oracle Clusterware can be implemented at two levels of availability for TimesTen. The *basic* level of availability manages two master nodes and up to 127 read-only subscriber nodes in the cluster. The active standby pair is defined with local host names or IP addresses. If both master nodes fail, user intervention is necessary to migrate the active standby scheme to new hosts. When both master nodes fail, Oracle Clusterware notifies the user.

The *advanced* level of availability uses virtual IP addresses for the active, standby and read-only subscriber databases. Extra nodes can be included in the cluster that are not part of the initial active standby pair. If a failure occurs, the use of virtual IP addresses allows one of the extra nodes to take on the role of a failed node automatically.

If your applications connect to TimesTen in a client/server configuration, automatic client failover enables the client to reconnect automatically to the master database with the active role after a failure. See "Using automatic client failover for an active standby pair" on page 3-13 and "TTC_FailoverPortRange" in the *Oracle TimesTen In-Memory Database Reference*.

The `ttCWAdmin` utility is used to administer TimesTen active standby pairs in a cluster that is managed by Oracle Clusterware. The configuration for each active standby pair is manually created in an initialization file called `cluster.oracle.ini` by default. The information in this file is used to create Oracle Clusterware *resources*. Resources are used to manage each TimesTen daemon, database, TimesTen processes, user applications and virtual IP addresses. For more information about the `ttCWAdmin` utility, see "ttCWAdmin" in *Oracle TimesTen In-Memory Database Reference*. For more information about the `cluster.oracle.ini` file, see "The cluster.oracle.ini file" on page 7-4.

Active standby configurations

Use Oracle Clusterware to manage only these configurations:

- Active standby pair with or without read-only subscribers
- Active standby pair (with or without read-only subscribers) with AWT cache groups, read-only cache groups and global cache groups

Required privileges

See "ttCWAdmin" in *Oracle TimesTen In-Memory Database Reference* for information about the privileges required to execute `ttCWAdmin` commands.

Hardware and software requirements

Oracle Clusterware release 11.2.0.2.*x* is supported with TimesTen active standby pair replication, beginning with release 11.2.0.2.0. See *Oracle Clusterware Administration and Deployment Guide* for network and storage requirements and information about Oracle Clusterware configuration files.

Oracle Clusterware and TimesTen should be installed in the same location on all nodes.

The TimesTen instance administrator must belong to the same UNIX primary group as the Oracle Clusterware installation owner.

Note that the `/tmp` directory contains essential TimesTen Oracle Clusterware directories. Their names have the prefix `crsTT`. Do not delete them.

All hosts should use Network Time Protocol (NTP) or a similar system so that clocks on the hosts remain within 250 milliseconds of each other.

Restricted commands and SQL statements

When you use Oracle Clusterware with TimesTen, you cannot use these commands and SQL statements:

- `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR` and `DROP ACTIVE STANDBY PAIR` SQL statements
- The `-repStart` and `-repStop` options of the `ttAdmin` utility
- The `-cacheStart` and `-cacheStop` options of the `ttAdmin` utility after the active standby pair has been created
- The `-duplicate` option of the `ttRepAdmin` utility
- The `ttRepStart` and `ttRepStop` built-in procedures
- Built-in procedures for managing a cache grid when the active standby pair in a cluster is a member of a grid

In addition, do not call `ttDaemonAdmin -stop` before calling `ttCWAdmin -shutdown`.

The TimesTen integration with Oracle Clusterware accomplishes these operations with the `ttCWAdmin` utility and the attributes in the `cluster.oracle.ini` file.

For more information about the built-ins and utilities, see *Oracle TimesTen In-Memory Database Reference*. For more information about the SQL statements, see *Oracle TimesTen In-Memory Database SQL Reference*.

The cluster.oracle.ini file

Create an initialization file called `cluster.oracle.ini` as a text file. The information in this file is used to create Oracle Clusterware resources that manage TimesTen databases, TimesTen processes, user applications and virtual IP addresses.

> **Note:** All of the attributes that can be used in the `cluster.oracle.ini` file are described in Chapter 8, "TimesTen Configuration Attributes for Oracle Clusterware".

The `ttCWAdmin -create` command reads this file for configuration information, so the location of the text file must be reachable by `ttCWAdmin`. It is recommended that you place this file in the daemon home directory on the host for the active database. However, you can place this file in any directory or shared drive on the same host as where you will execute the `ttCWAdmin -create` command.

The default location for this file is in one of the following directories:

- The *`install_dir`*`/info` directory on UNIX platforms
- The `c:\TimesTen\`*`install_dir`*`\srv\info` directory on Windows platforms

If you place this file in another location, identify the path of the location with the `-ttclusterini` option.

The entry name in the `cluster.oracle.ini` file must be the same as an existing DSN:

- In the `sys.odbc.ini` file on UNIX platforms
- In a system DSN on Windows platforms

For example, `[basicDSN]` is the entry name in the `cluster.oracle.ini` file described in "Configuring basic availability" on page 7-5. `[basicDSN]` must also be the `DataStore` and `Data Source Name` data store attributes in the `sys.odbc.ini` files on each host. For example, the `sys.odbc.ini` file for the `basicDSN` DSN on `host1` might be:

```
[basicDSN]
DataStore=/path1/basicDSN
LogDir=/path1/log
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

The `sys.odbc.ini` file for `basicDSN` on `host2` can have a different path, but all other attributes should be the same:

```
[basicDSN]
DataStore=/path2/basicDSN
LogDir=/path2/log
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

This section includes sample `cluster.oracle.ini` files for these configurations:

- Configuring basic availability
- Configuring advanced availability
- Including cache groups in the active standby pair
- Including the active standby pair in a cache grid

- Implementing application failover
- Recovering from permanent failure of both master nodes
- Using the RepDDL attribute

Configuring basic availability

This example shows an active standby pair with no subscribers. The hosts for the active database and the standby database are `host1` and `host2`. The list of hosts is delimited by commas. You can include spaces for readability if desired.

The `ttCWAdmin` utility is used to administer TimesTen active standby pairs in a cluster that is managed by Oracle Clusterware.

```
[basicDSN]
MasterHosts=host1,host2
```

The following is an example of a `cluster.oracle.ini` file for an active standby pair with one subscriber on `host3`:

```
[basicSubscriberDSN]
MasterHosts=host1,host2
SubscriberHosts=host3
```

Configuring advanced availability

In this example, the hosts for the active database and the standby database are `host1` and `host2`. The specified `host3` and `host4` are extra nodes that can be used for failover. There are no subscriber nodes. `MasterVIP` specifies the virtual IP addresses defined for the master databases. `VIPInterface` is the name of the public network adaptor. `VIPNetMask` defines the netmask of the virtual IP addresses.

```
[advancedDSN]
MasterHosts=host1,host2,host3,host4
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0
```

This example has one subscriber on `host4`. There is one extra node that can be used for failing over the master databases and one extra node that can be used for the subscriber database. `MasterVIP` and `SubscriberVIP` specify the virtual IP addresses defined for the master and subscriber databases. `VIPInterface` is the name of the public network adaptor. `VIPNetMask` defines the netmask of the virtual IP addresses.

```
[advancedSubscriberDSN]
MasterHosts=host1,host2,host3
SubscriberHosts=host4,host5
MasterVIP=192.168.1.1, 192.168.1.2
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0
```

Ensure that the extra nodes:

- Have TimesTen installed
- Have the direct-linked application installed if this is part of the configuration. See "Implementing application failover" on page 7-6.

Including cache groups in the active standby pair

If the active standby pair replicates one or more AWT or read-only cache groups, set the CacheConnect attribute to y.

This example specifies an active standby pair with one subscriber in an advanced availability configuration. The active standby pair replicates one or more cache groups.

```
[advancedCacheDSN]
MasterHosts=host1,host2,host3
SubscriberHosts=host4, host5
MasterVIP=192.168.1.1, 192.168.1.2
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0
CacheConnect=y
```

Including the active standby pair in a cache grid

If the active standby pair is a member of a cache grid, assign port numbers for the active and standby databases by setting the GridPort attribute.

This example specifies an active standby pair with no subscribers in an advanced availability configuration. The active standby pair is a member of a cache grid.

```
[advancedGridDSN]
MasterHosts=host1,host2,host3
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0
CacheConnect=y
GridPort=16101, 16102
```

For more information about using Oracle Clusterware with a cache grid, see "Using Oracle Clusterware with a TimesTen cache grid" on page 7-15.

Implementing application failover

TimesTen integration with Oracle Clusterware can facilitate the failover of a TimesTen application that is linked to any of the databases in the active standby pair. Both direct-linked and client/server applications that are on the same host as Oracle Clusterware and TimesTen can be managed.

The required attributes in the `cluster.oracle.ini` file for failing over a TimesTen application are:

- AppName - Name of the application to be managed by Oracle Clusterware
- AppStartCmd - Command line for starting the application
- AppStopCmd - Command line for stopping the application
- AppCheckCmd - Command line for executing an application that checks the status of the application specified by AppName
- AppType - Determines the database to which the application is linked. The possible values are Active, Standby, DualMaster, Subscriber (all) and Subscriber[*index*].

Optionally, you can also set AppFailureThreshold, DatabaseFailoverDelay, and AppScriptTimeout. These attributes have default values.

The TimesTen application monitor process uses the user-supplied script or program specified by AppCheckCmd to monitor the application. The script that checks the status of the application must be written to return 0 for success and a nonzero number for failure. When Oracle Clusterware detects a nonzero value, it takes action to recover the failed application.

This example shows advanced availability configured for an active standby pair with with no subscribers. The reader application is an application that queries the data in the standby database. AppStartCmd, AppStopCmd and AppCheckCmd can include arguments such as start, stop and check commands. On UNIX, do not use quotes in the values for AppStartCmd, AppStopCmd and AppCheckCmd.

```
[appDSN]
MasterHosts=host1,host2,host3,host4
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0
AppName=reader
AppType=Standby
AppStartCmd=/mycluster/reader/app_start.sh start
AppStopCmd=/mycluster/reader/app_stop.sh stop
AppCheckCmd=/mycluster/reader/app_check.sh check
```

AppStartCmd, AppStopCmd and AppCheckCmd can include arguments. For example, the following is a valid cluster.oracle.ini file on Windows that demonstrates configuration for an application that is directly linked to the active database. The script for starting, stopping, and checking the application takes arguments for the DSN and the action to take (-start, -stop and -check).

Note the double quotes for the specified paths in AppStartCmd, AppStopCmd and AppCheckCmd. The quotes are needed because there are spaces in the path. Enclose only the path in quotes. Do not enclose the DSN or the action in quotes.

```
[appWinDSN]
MasterHosts=host1,host2,host3,host4
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=Local Area Connection
VIPNetMask=255.255.255.0
AppName=UpdateApp
AppType=Active
AppStartCmd="C:\Program Files\UserApps\UpdateApp.exe" -dsn myDSN -start
AppStopCmd= "C:\Program Files\UserApps\UpdateApp.exe" -dsn myDSN -stop
AppCheckCmd="C:\Program Files\UserApps\UpdateApp.exe" -dsn myDSN -check
```

You can configure failover for more than one application. Use AppName to name the application and provide values for AppType, AppStartCmd, AppStopCmd and AppCheckCmd immediately following the AppName attribute. You can include blank lines for readability. For example:

```
[app2DSN]
MasterHosts=host1,host2,host3,host4
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0

AppName=reader
AppType=Standby
AppStartCmd=/mycluster/reader/app_start.sh
AppStopCmd=/mycluster/reader/app_stop.sh
AppCheckCmd=/mycluster/reader/app_check.sh
```

```
AppName=update
AppType=Active
AppStartCmd=/mycluster/update/app2_start.sh
AppStopCmd=/mycluster/update/app2_stop.sh
AppCheckCmd=/mycluster/update/app2_check.sh
```

The application is considered available if it has been running for 15 times the value of `AppScriptTimeout` attribute. The default value of `AppScriptTimeout` is 60 seconds, so the application's "uptime threshold" is 15 minutes by default. If the application fails after running for more than 15 minutes, it will be restarted on the same host. If the application fails within 15 minutes of being started, the failure is considered a failure to start properly, and the application will be restarted on another host. If you want to modify the application's uptime threshold after the application has started, use the `crs_register -update` command. See *Oracle Clusterware Administration and Deployment Guide* for information about the `crs_register -update` command.

If you set AppType to `DualMaster`, the application starts on both the active host and the standby host.The failure of the application on the active host causes the active database and all other applications on the host to fail over to the standby host. You can configure the failure interval, the number of restart attempts and the uptime threshold by setting the `AppFailureInterval`, `AppRestartAttempts` and `AppUptimeThreshold` attributes. These attributes have default values. For example:

```
[appDualDSN]
MasterHosts=host1,host2,host3,host4
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0
AppName=update
AppType=DualMaster
AppStartCmd=/mycluster/update/app2_start.sh
AppStopCmd=/mycluster/update/app2_stop.sh
AppCheckCmd=/mycluster/update/app2_check.sh
AppRestartAttempts=5
AppUptimeThreshold=300
AppFailureInterval=30
```

Recovering from permanent failure of both master nodes

If both master nodes fail and then come back up, Oracle Clusterware can automatically recover the master databases. Automatic recovery of temporary dual failure requires:

- `RETURN TWOSAFE` is not specified for the active standby pair.
- `AutoRecover` is set to `y`.
- `RepBackupDir` specifies a directory on shared storage.
- `RepBackupPeriod` is set to a value greater than `0`.

If both master nodes fail permanently, Oracle Clusterware can automatically recover the master databases to two new nodes if:

- Advanced availability is configured (virtual IP addresses and at least four hosts).
- The active standby pair does not replicate cache groups.
- A cache grid is not configured.
- `RETURN TWOSAFE` is not specified.

- AutoRecover is set to y.
- RepBackupDir specifies a directory on shared storage.
- RepBackupPeriod must be set to a value greater than 0.

TimesTen first performs a full backup of the active database and then performs incremental backups. You can specify the optional attribute RepFullBackupCycle to manage when TimesTen performs subsequent full backup. By default, TimesTen performs a full backup after every five incremental backups.

If RepBackupDir and RepBackupPeriod are configured for backups, TimesTen performs backups for any master database that becomes active. It does not delete backups that were performed for a database that used to be active and has become the standby unless the database becomes active again. Ensure that the shared storage has enough space for two complete database backups. ttCWAdmin -restore automatically chooses the correct backup files.

Incremental backups increase the amount of log records in the transaction log files. Ensure that the values of RepBackupPeriod and RepFullBackupCycle are small enough to prevent a large amount of log records in the transaction log file.

This example shows attribute settings for automatic recovery.

```
[autorecoveryDSN]
MasterHosts=host1,host2,host3,host4
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0
AutoRecover=y
RepBackupDir=/shared_drive/dsbackup
RepBackupPeriod=3600
```

If you have cache groups in the active standby pair or prefer to recover manually from failure of both master hosts, ensure that AutoRecover is set to n (the default). Manual recovery requires:

- RepBackupDir specifies a directory on shared storage
- RepBackupPeriod must be set to a value greater than 0

This example shows attribute settings for manual recovery. The default value for AutoRecover is n, so it is not included in the file.

```
[manrecoveryDSN]
MasterHosts=host1,host2,host3
MasterVIP=192.168.1.1, 192.168.1.2
VIPInterface=eth0
VIPNetMask=255.255.255.0
RepBackupDir=/shared_drive/dsbackup
RepBackupPeriod=3600
```

Using the RepDDL attribute

The RepDDL attribute represents the SQL statement that creates the active standby pair. The RepDDL attribute is optional. You can use it to exclude tables, cache groups and sequences from the active standby pair.

If you include RepDDL in the cluster.oracle.ini file, do not specify ReturnServiceAttribute, MasterStoreAttribute or SubscriberStoreAttribute in the cluster.oracle.ini file. Include those replication settings in the RepDDL attribute.

When you specify a value for RepDDL, use the <DSN> macro for the database file name prefix. Use the <MASTERHOST[1]> and <MASTERHOST[2]> macros to specify the master host names. TimesTen substitutes the correct values from the MasterHosts or MasterVIP attributes, depending on whether your configuration uses virtual IP addresses. Similarly, use the <SUBSCRIBERHOST[*n*]> macro to specify subscriber host names, where *n* is a number from 1 to the total number of SubscriberHosts attribute values or 1 to the total number of SubscriberVIP attribute values if virtual IP addresses are used.

Use the RepDDL attribute to exclude tables, cache groups and sequences from the active standby pair:

```
[excludeDSN]
MasterHosts=host1,host2,host3,host4
SubscriberHosts=host5,host6
MasterVIP=192.168.1.1, 192.168.1.2
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0
RepDDL=CREATE ACTIVE STANDBY PAIR \
<DSN> ON <MASTERHOST[1]>, <DSN> ON <MASTERHOST[2]>
SUBSCRIBER <DSN> ON <SUBSCRIBERHOST[1]>\
EXCLUDE TABLE pat.salaries, \
EXCLUDE CACHE GROUP terry.salupdate, \
EXCLUDE SEQUENCE ttuser.empcount
```

The replication agent transmitter obtains route information as follows, in order of priority:

1. From the ROUTE clause in the RepDDL setting, if a ROUTE clause is specified. Do not specify a ROUTE clause if you are configuring advanced availability.
2. From Oracle Clusterware, which provides the private host names and public host names of the local and remote hosts as well as the remote daemon port number. The private host name is preferred over the public host name. The replication agent transmitter cannot connect to the IPC socket, it attempts to connect to the remote daemon, using information that Oracle Clusterware maintains about the replication scheme.
3. From the active and standby hosts. If they fail, then the replication agent chooses the connection method based on host name.

This is an example of specifying the ROUTE clause in RepDDL:

```
[routeDSN]
MasterHosts=host1,host2,host3,host4
RepDDL=CREATE ACTIVE STANDBY PAIR \
<DSN> ON <MASTERHOST[1]>, <DSN> ON <MASTERHOST[2]>\
ROUTE MASTER <DSN> ON <MASTERHOST[1]>  SUBSCRIBER <DSN> ON <MASTERHOST[2]>\
MASTERIP "192.168.1.2" PRIORITY 1\
SUBSCRIBERIP "192.168.1.3" PRIORITY 1\
MASTERIP "10.0.0.1" PRIORITY 2\
SUBSCRIBERIP "10.0.0.2" PRIORITY 2\
MASTERIP "140.87.11.203" PRIORITY 3\
SUBSCRIBERIP "140.87.11.204" PRIORITY 3\
ROUTE MASTER <DSN> ON <MASTERHOST[2]>  SUBSCRIBER <DSN> ON <MASTERHOST[1]>\
MASTERIP "192.168.1.3" PRIORITY 1\
SUBSCRIBERIP "192.168.1.2" PRIORITY 1\
MASTERIP "10.0.0.2" PRIORITY 2\
SUBSCRIBERIP "10.0.0.1" PRIORITY 2\
MASTERIP "140.87.11.204" PRIORITY 3\
```

```
SUBSCRIBERIP "140.87.11.203" PRIORITY 3\
```

Creating and initializing a cluster

To create and initialize a cluster, perform these tasks:

- Install Oracle Clusterware
- Install TimesTen on each host
- Register the TimesTen cluster information
- Start the TimesTen cluster agent
- Create and populate a TimesTen database on one host
- Create sys.odbc.ini files on other hosts
- Create a cluster.oracle.ini file
- Create the virtual IP addresses (optional)
- Create an active standby pair replication scheme
- Start the active standby pair
- Load cache groups

If you plan to have more than one active standby pair in the cluster, see "Including more than one active standby pair in a cluster" on page 7-14.

If you want to configure an Oracle database as a remote disaster recovery subscriber, see "Configuring an Oracle database as a disaster recovery subscriber" on page 7-14.

If you want to set up a read-only subscriber that is not managed by Oracle Clusterware, see "Configuring a read-only subscriber that is not managed by Oracle Clusterware" on page 7-15.

Install Oracle Clusterware

Install Oracle Clusterware. By default, the installation occurs on all hosts concurrently. See Oracle Clusterware installation documentation for your platform.

Oracle Clusterware starts automatically after successful installation.

Install TimesTen on each host

Install TimesTen in the same location on each host in the cluster, including extra hosts. The instance name must be the same on each host. The user name of the instance administrator must be the same on all hosts. The TimesTen instance administrator must belong to the same UNIX primary group as the Oracle Clusterware installation owner.

On UNIX platforms, the installer prompts you for values for:

- The TCP/IP port number associated with the TimesTen cluster agent. The port number can be different on each host. If you do not provide a port number, TimesTen uses the default TimesTen port.
- The Oracle Clusterware location. The location must be the same on each host.
- The hosts included in the cluster, including spare hosts, with host names separated by commas. This list must be the same on each host.

The installer uses these values to create the `ttcrsagent.options` file on UNIX platforms. See "TimesTen Installation" in *Oracle TimesTen In-Memory Database Installation Guide* for details. You can also use `ttmodinstall -crs` to create the file after installation. Use the `-record` and `-batch` options for `setup.sh` to perform identical installations on additional hosts if desired.

On Windows, execute `ttmodinstall -crs` on each node after installation to create the `ttcrsagent.options` file.

For more information about `ttmodinstall`, see "ttmodinstall" in *Oracle TimesTen In-Memory Database Reference*.

Register the TimesTen cluster information

TimesTen cluster information is stored in the Oracle Cluster Registry (OCR). As the root user on UNIX platforms, or as the instance administrator on Windows, enter this command:

```
ttCWAdmin -ocrConfig
```

As long as Oracle Clusterware and TimesTen are installed on the hosts, this step never needs to be repeated.

Start the TimesTen cluster agent

Start the TimesTen cluster agent by executing the `ttCWAdmin -init` command on one of the hosts. For example:

```
ttCWAdmin -init
```

This command starts the TimesTen cluster agent (`ttCRSAgent`) and the TimesTen daemon monitor (`ttCRSDaemon`). There is one TimesTen cluster agent and one TimesTen daemon monitor for the TimesTen installation. When the TimesTen cluster agent has started, Oracle Clusterware begins monitoring the TimesTen daemon and will restart it if it fails.

Note: You must stop the TimesTen cluster agent on the local host before you execute a `ttDaemonAdmin -stop` command. Otherwise the cluster agent will restart the daemon.

Create and populate a TimesTen database on one host

Create a database on the host where you intend the active database to reside. The DSN must be the same as the database file name.

Create schema objects such as tables, AWT cache groups and read-only cache groups. Do not load the cache groups.

Create sys.odbc.ini files on other hosts

On all hosts that will be in the cluster, create `sys.odbc.ini` files. The `DataStore` attribute and the `Data Source Name` must be the same as the entry name for the `cluster.oracle.ini` file. See "The cluster.oracle.ini file" on page 7-4 for information about the contents of the `sys.odbc.ini` files.

Create a cluster.oracle.ini file

Create a `cluster.oracle.ini` file as a text file. See "The cluster.oracle.ini file" on page 7-4 for details about its contents and acceptable locations for the file.

Create the virtual IP addresses (optional)

For advanced availability, execute the `ttCWAdmin -createVIPs` command on any host in the cluster. On UNIX, you must execute this command as the `root` user. For example:

```
ttCWAdmin -createVIPs -dsn myDSN
```

Create an active standby pair replication scheme

Create an active standby pair replication scheme by executing the `ttCWAdmin -create` command on any host.

> **Note:** The `cluster.oracle.ini` file contains the configuration needed to perform the `ttCWAdmin -create` command and so must reachable by the `ttCWAdmin` executable. See "The cluster.oracle.ini file" on page 7-4 for details about acceptable locations for the `cluster.oracle.ini` file.

For example:

```
ttCWAdmin -create -dsn myDSN
```

This command prompts for an encryption pass phrase that the user will not need again. The command also prompts for the user ID and password for an internal user with the `ADMIN` privilege if it does not find this information in the `sys.odbc.ini` file. This internal user will be used to create the active standby pair.

If the `CacheConnect` Clusterware attribute is enabled, the command prompts for the user password for the Oracle database. The Oracle password is used to set the autorefresh states for cache groups. See "CacheConnect" on page 8-16 for more details on this attribute.

Start the active standby pair

Start the active standby pair replication scheme by executing the `ttCWAdmin -start` command on any host. For example:

```
ttCWAdmin -start -dsn myDSN
```

This command starts the following processes for the active standby pair:

- `ttCRSMaster`
- `ttCRSActiveService`
- `ttCRSsubservice`
- Monitor for application `AppName`

Load cache groups

If the active standby pair includes cache groups, use the `LOAD CACHE GROUP` statement to load the cache group tables from the Oracle tables.

Including more than one active standby pair in a cluster

If you want to use Oracle Clusterware to manage more than one active standby pair in a cluster, include additional configuration in the `cluster.oracle.ini` file. Oracle Clusterware can only manage more than one active standby pair in a cluster if all TimesTen databases are a part of the same TimesTen instance on a single host.

For example, the following `cluster.oracle.ini` file contains configuration information for two active standby pair replication schemes on the same host:

> **Note:** For details on configuration attributes in the `cluster.oracle.ini` file, see Chapter 8, "TimesTen Configuration Attributes for Oracle Clusterware".

```
[advancedSubscriberDSN]
MasterHosts=host1,host2,host3
SubscriberHosts=host4, host5
MasterVIP=192.168.1.1, 192.168.1.2
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0

[advSub2DSN]
MasterHosts=host1,host2,host3
SubscriberHosts=host4, host5
MasterVIP=192.168.1.4, 192.168.1.5
SubscriberVIP=192.168.1.6
VIPInterface=eth0
VIPNetMask=255.255.255.0
```

Perform these tasks for additional replication schemes:

1. Create and populate the databases.
2. Create the virtual IP addresses. Use the `ttCWAdmin -createVIPs` command.
3. Create the active standby pair replication scheme. Use the `ttCWAdmin -create` command.
4. Start the active standby pair. Use the `ttCWAdmin -start` command.

Configuring an Oracle database as a disaster recovery subscriber

You can create an active standby pair on the primary site with an Oracle database as a remote disaster recovery subscriber. See "Using a disaster recovery subscriber in an active standby pair" on page 5-9. Oracle Clusterware manages the active standby pair but does not manage the disaster recovery subscriber. The user must perform a switchover if the primary site fails.

To use Oracle Clusterware to manage an active standby pair that has a remote disaster recovery subscriber, perform these tasks:

1. Use the `RepDDL` or `RemoteSubscriberHosts` Clusterware attribute to provide information about the remote disaster recovery subscriber.For example:

```
[advancedDRsubDSN]
MasterHosts=host1,host2,host3
SubscriberHosts=host4, host5
RemoteSubscriberHosts=host6
MasterVIP=192.168.1.1, 192.168.1.2
```

```
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0
CacheConnect=y
```

2. Use `ttCWAdmin -create` to create the active standby pair replication scheme on the primary site. This does not create the disaster recovery subscriber.
3. Use `ttCWAdmin -start` to start the active standby pair replication scheme.
4. Load the cache groups that are replicated by the active standby pair.
5. Set up the disaster recovery subscriber using the procedure in "Rolling out a disaster recovery subscriber" on page 5-10.

Configuring a read-only subscriber that is not managed by Oracle Clusterware

You can include a read-only TimesTen subscriber database that is not managed by Oracle Clusterware. Perform these tasks:

1. Include the RemoteSubscriberHosts Clusterware attribute in the `cluster.oracle.ini` file.For example:

```
[advancedROsubDSN]
MasterHosts=host1,host2,host3
RemoteSubscriberHosts=host6
MasterVIP=192.168.1.1, 192.168.1.2
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0
```

2. Use `ttCWAdmin -create` to create the active standby pair replication scheme on the primary site.
3. Use `ttCWAdmin -start` to start the active standby pair replication scheme. This does not create the read-only subscriber.
4. Use the `ttRepStateGet` procedure to verify that the state of the standby database is `STANDBY`.
5. On the subscriber host, use `ttRepAdmin -duplicate` option to duplicate the standby database to the read-only subscriber. See "Duplicating a database" on page 4-2.
6. Start the replication agent on the subscriber host.

To add a read-only subscriber to an existing configuration, see "Adding a read-only subscriber not managed by Oracle Clusterware" on page 7-28.

To rebuild a read-only subscriber, see "Rebuilding a read-only subscriber not managed by Oracle Clusterware" on page 7-29.

Using Oracle Clusterware with a TimesTen cache grid

You can use the TimesTen implementation of Oracle Clusterware to manage a cache grid when each grid member is an active standby pair. TimesTen does not support using Oracle Clusterware to manage standalone grid members.

This section includes:

- Creating and initializing a cluster of cache grid members
- Failure and recovery for active standby pair grid members

- Making schema changes to active standby pairs in a grid

Creating and initializing a cluster of cache grid members

See "Install TimesTen on each host" on page 7-11 for installation requirements. In addition, each grid member must have a DSN that is unique within the cache grid.

Perform the tasks described in "Creating and initializing a cluster" on page 7-11 for each grid member. Include the GridPort Clusterware attribute in the `cluster.oracle.ini` file as described in "Including the active standby pair in a cache grid" on page 7-6. Ensure that the specified port numbers are not in use.

The `ttCWAdmin -start` command automatically attaches a grid member to the cache grid attach. The `ttCWAdmin -stop` command automatically detaches a grid member from the cache grid.

Failure and recovery for active standby pair grid members

If both nodes of an active standby pair grid member fail, then the grid member fails. Oracle Clusterware evicts the failed grid member from the grid automatically. However, when a cache grid is configured, any further automatic recovery after a dual failure, whether temporary or permanent, is not possible. In this case, you can only recover manually. For details, see "Manual recovery of both nodes of an active standby pair grid member" on page 7-22.

Making schema changes to active standby pairs in a grid

You can add, drop or change a cache group while the active database is attached to the grid.

Use the `ttCWAdmin -beginAlterSchema` command to make these schema changes. This command stops replication but allows the active database to remain attached to the grid. The `ttCWAdmin -endAlterSchema` command duplicates the changes to the standby database, registers the altered replication scheme and starts replication.

To add a table and include it in the active standby pair, see "Making DDL changes in an active standby pair" on page 6-1. See the same section for information about dropping a replicated table.

Add a cache group

Perform these steps on the active database of *each* active standby pair grid member.

1. Enable the addition of the cache group to the active standby pair.

   ```
   ttCWAdmin -beginAlterSchema advancedGridDSN
   ```

2. Create the cache group.
3. If the cache group is a read-only cache group, alter the active standby pair to include the cache group.

   ```
   ALTER ACTIVE STANDBY PAIR INCLUDE CACHE GROUP samplecachegroup;
   ```

4. Duplicate the change to the standby database.

   ```
   ttCWAdmin -endAlterSchema advancedGridDSN
   ```

You can load the cache group at any time after you create the cache group.

Drop a cache group

Perform these steps to drop a cache group.

1. Unload the cache group in all members of the cache grid.

```
CALL ttOptSetFlag('GlobalProcessing', 1);
UNLOAD CACHE GROUP samplecachegroup;
```

2. On the active database of an active standby pair grid member, enable dropping the cache group.

```
ttCWAdmin -beginAlterSchema advancedGridDSN
```

3. If the cache group is a read-only cache group, alter the active standby pair to exclude the cache group.

```
ALTER ACTIVE STANDBY PAIR EXCLUDE CACHE GROUP samplecachegroup;
```

4. If the cache group is a read-only cache group, set the autorefresh state to `PAUSED`.

```
ALTER CACHE GROUP samplecachegroup SET AUTOREFRESH STATE PAUSED;
```

5. Drop the cache group.

```
DROP CACHE GROUP samplecachegroup;
```

6. If the cache group was a read-only cache group, run the *`TimesTen_install_dir`*`/oraclescripts/cacheCleanUp.sql` SQL*Plus script as the cache administration user on the Oracle database to drop the Oracle objects used to implement autorefresh operations.
7. Duplicate the change to the standby database.

```
ttCWAdmin -endAlterSchema advancedGridDSN
```

8. Repeat steps 2 through 7 on the active database of each active standby pair grid member.

Change an existing cache group

To change an existing cache group, first drop the existing cache group as described in "Drop a cache group" on page 7-17. Then add the cache group with the desired changes as described in "Add a cache group" on page 7-16.

Recovering from failures

Oracle Clusterware can recover automatically from many kinds of failures. The following sections describe several failure scenarios and how Oracle Clusterware manages the failures.

- How TimesTen performs recovery when Oracle Clusterware is configured
- When an active database or its host fails
- When a standby database or its host fails
- When read-only subscribers or their hosts fail
- When failures occur on both master nodes
- When more than two master hosts fail
- Performing a forced switchover after failure of the active database or host

How TimesTen performs recovery when Oracle Clusterware is configured

The TimesTen database monitor (`ttCRSmaster` process) performs recovery. It attempts to connect to the failed database without using the `forceconnect` option. If the connection fails with error 994 (`Data store connection terminated`), the database monitor tries to connect 10 times. If the connection fails with error 707 (`Attempt to connect to a data store that has been manually unloaded from RAM`), the database monitor changes the RAM policy and tries to connect again. If the database monitor cannot connect, it returns connection failure.

If the database monitor can connect to the database, then it performs these tasks:

- It queries the `CHECKSUM` column in the `TTREP.REPLICATIONS` replication table.
- If the value in the `CHECKSUM` column matches the checksum stored in the Oracle Cluster Registry, then the database monitor verifies the role of the database. If the role is `'ACTIVE'`, then recovery is complete.

  If the role is not `'ACTIVE'`, then the database monitor queries the replication Commit Ticket Number (CTN) in the local database and the CTN in the active database to find out whether there are transactions that have not been replicated. If all transactions have been replicated, then recovery is complete.
- If the checksum does not match or if some transactions have not been replicated, then the database monitor performs a duplicate operation from the remote database to re-create the local database.

If the database monitor fails to connect with the database because of error 8110 or 8111 (master catchup required or in progress), then it uses the `forceconnect=1` option to connect and starts master catchup. Recovery is complete when master catchup has been completed. If master catchup fails with error 8112 (`Operation not permitted`), then the database monitor performs a duplicate operation from the remote database. For more information about master catchup, see "Automatic catch-up of a failed master database" on page 11-3.

If the connection fails because of other errors, then the database monitor tries to perform a duplicate operation from the remote database.

The duplicate operation verifies that:

- The remote database is available.
- The replication agent is running.
- The remote database has the correct role. The role must be `'ACTIVE'` when the duplicate operation is attempted for creation of a standby database. The role must be `'STANDBY'` or `'ACTIVE'` when the duplicate operation is attempted for creation of a read-only subscriber.

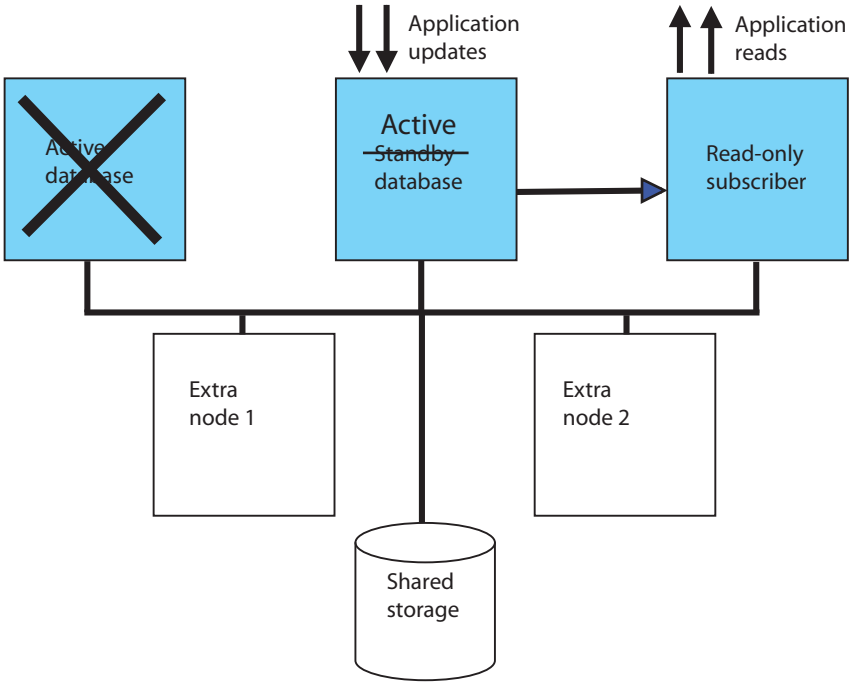
When the conditions for the duplicate operation are satisfied, the existing failed database is destroyed and the duplicate operation starts.

When an active database or its host fails

If there is a failure on the node where the active database resides, Oracle Clusterware automatically changes the state of the standby database to `'ACTIVE'`. If application failover is configured, then the application begins updating the new active database.

Figure 7–2 shows that the state of the standby database has changed to `'ACTIVE'` and that the application is updating the new active database.

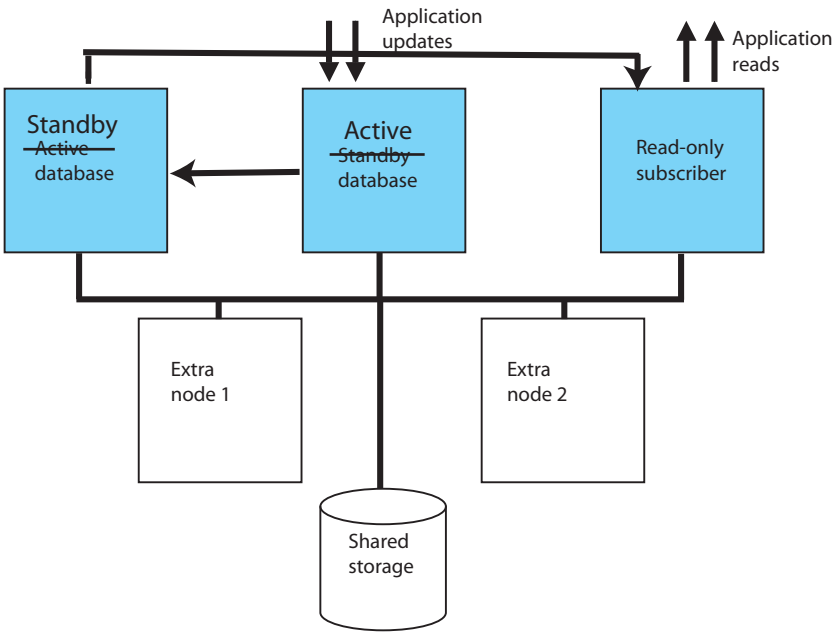
Figure 7–2 Standby database becomes active



Oracle Clusterware tries to restart the database or host where the failure occurred. If it is successful, then that database becomes the standby database.

Figure 7–3 shows a cluster where the former active node becomes the standby node.

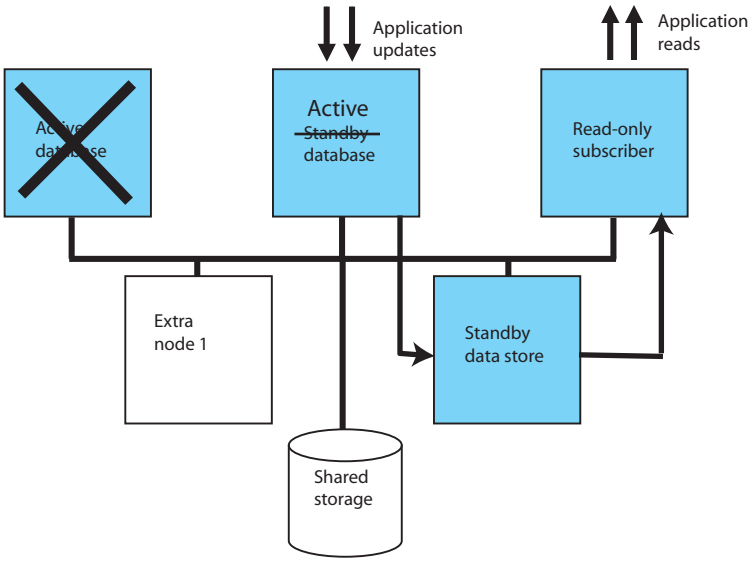
Figure 7–3 Standby database starts on former active host



If the failure of the former active node is permanent and advanced availability is configured, Oracle Clusterware starts a standby database on one of the extra nodes.

Figure 7–4 shows a cluster in which the standby database is started on one of the extra nodes.

Figure 7–4 Standby database starts on extra host



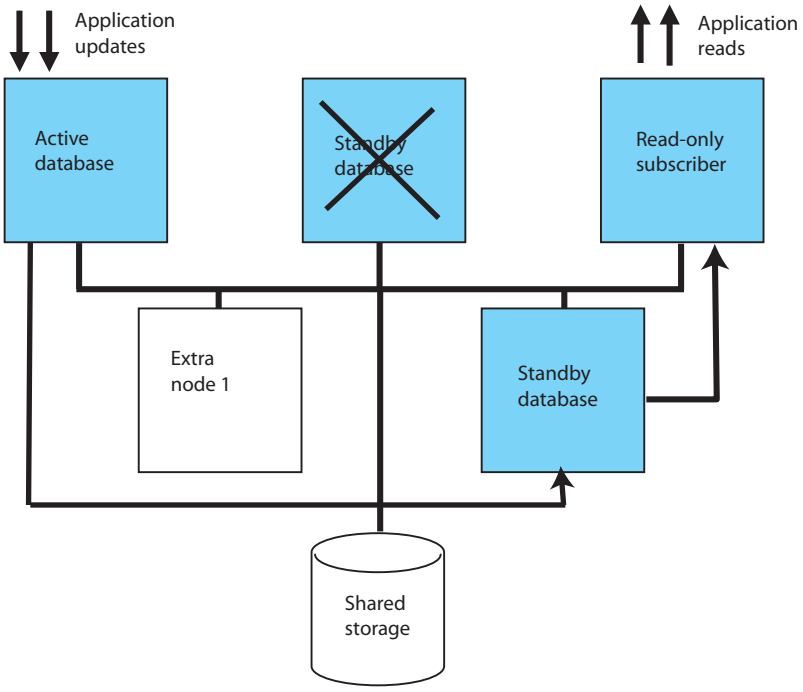
If you do not want to wait for these automatic actions to occur, see "Performing a forced switchover after failure of the active database or host" on page 7-25.

When a standby database or its host fails

If there is a failure on the standby node, Oracle Clusterware first tries to restart the database or host. If it cannot restart the standby database on the same host and advanced availability is configured, Oracle Clusterware starts the standby database on an extra node.

Figure 7–5 shows a cluster in which the standby database is started on one of the extra nodes.

Figure 7–5 Standby database on new host



When read-only subscribers or their hosts fail

If there is a failure on a subscriber node, Oracle Clusterware first tries to restart the database or host. If it cannot restart the database on the same host and advanced availability is configured, Oracle Clusterware starts the subscriber database on an extra node.

When failures occur on both master nodes

This section includes these topics:

- Automatic recovery when not attached to a grid
- Manual recovery of both nodes of an active standby pair grid member
- Manual recovery for advanced availability
- Manual recovery for basic availability
- Manual recovery to the same master nodes when databases are corrupt
- Manual recovery when RETURN TWOSAFE is configured

Automatic recovery when not attached to a grid

Oracle Clusterware can achieve automatic recovery from *temporary* failure on both master nodes after the nodes come back up if:

- `RETURN TWOSAFE` is not specified for the active standby pair.
- `AutoRecover` is set to `y`.
- `RepBackupDir` specifies a directory on shared storage.
- `RepBackupPeriod` is set to a value greater than `0`.

Oracle Clusterware can achieve automatic recovery from *permanent* failure on both master nodes if:

- Advanced availability is configured (virtual IP addresses and at least four hosts).
- The active standby pair does not replicate cache groups.
- A cache grid is not configured.
- `RETURN TWOSAFE` is not specified for the active standby pair.
- `AutoRecover` is set to `y`.
- `RepBackupDir` specifies a directory on shared storage.
- `RepBackupPeriod` is set to a value greater than `0`.

See "Recovering from permanent failure of both master nodes" on page 7-8 for examples of `cluster.oracle.ini` files.

Manual recovery of both nodes of an active standby pair grid member

If both nodes of an active standby pair grid member fail, then the grid member fails. Oracle Clusterware evicts the failed grid member from the grid automatically. After the failed grid member is removed from the grid, you can continue to recover manually. However, when a cache grid is configured, any further automatic recovery after a dual failure, whether temporary or permanent, is not possible.

If the active standby pair grid member is in an asynchronous replication scheme, the grid member is recovered automatically and reattached to the grid. If the active standby pair grid member is in a replication scheme with `RETURN TWOSAFE` configured, perform these steps to recover the grid member and reattach it to the grid:

1. Stop the replication agent and the cache agent and disconnect the application from both databases. This step detaches the grid member from the grid.

   ```
   ttCWAdmin -stop advancedGridDSN
   ```

2. Drop the active standby pair.

   ```
   ttCWAdmin -drop advancedGridDSN
   ```

3. Create the active standby pair replication scheme.

   ```
   ttCWAdmin -create advancedGridDSN
   ```

4. Start the active standby pair replication scheme. This step attaches the grid member to the grid.

   ```
   ttCWAdmin -start advancedGridDSN
   ```

Manual recovery for advanced availability

This section assumes that the failed master nodes will be recovered to new hosts on which TimesTen and Oracle Clusterware have been installed. These steps use the `manrecoveryDSN` database and `cluster.oracle.ini` file for examples.

To perform manual recovery in an advanced availability configuration, perform these tasks:

1. Ensure that the TimesTen cluster agent is running on the local host.

   ```
   ttCWAdmin -init -hosts localhost
   ```

2. Restore the backup database. Ensure that there is not already a database on the host with the same DSN as the database you want to restore.

```
ttCWAdmin -restore -dsn manrecoveryDSN
```

3. If there are cache groups in the database, drop and re-create the cache groups.
4. If the new hosts are not already specified by `MasterHosts` and `SubscriberHosts` in the `cluster.oracle.ini` file, then modify the file to include the new hosts.

   These steps use `manrecoveryDSN`. This step is not necessary for `manrecoveryDSN` because extra hosts are already specified in the `cluster.oracle.ini` file.
5. Re-create the active standby pair replication scheme.

```
ttCWAdmin -create -dsn manrecoveryDSN
```

6. Start the active standby pair replication scheme.

```
ttCWAdmin -start -dsn manrecoveryDSN
```

Manual recovery for basic availability

This section assumes that the failed master nodes will be recovered to new hosts on which TimesTen and Oracle Clusterware have been installed. These steps use the `basicDSN` database and `cluster.oracle.ini` file for examples.

To perform manual recovery in a basic availability configuration, perform these steps:

1. Acquire new hosts for the databases in the active standby pair.
2. Ensure that the TimesTen cluster agent is running on the local host.

```
ttCWAdmin -init -hosts localhost
```

3. Restore the backup database. Ensure that there is not already a database on the host with the same DSN as the database you want to restore.

```
ttCWADmin -restore -dsn basicDSN
```

4. If there are cache groups in the database, drop and re-create the cache groups.
5. Update the `MasterHosts` and `SubscriberHosts` entries in the `cluster.oracle.ini` file. This example uses the `basicDSN` database. The `MasterHosts` entry changes from `host1` to `host10`. The `SubscriberHosts` entry changes from `host2` to `host20`.

```
[basicDSN]
MasterHosts=host10,host20
```

6. Re-create the active standby pair replication scheme.

```
ttCWAdmin -create -dsn basicDSN
```

7. Start the active standby pair replication scheme.

```
ttCWAdmin -start -dsn basicDSN
```

Manual recovery to the same master nodes when databases are corrupt

Failures can occur on both master nodes so that the databases are corrupt. If you want to recover to the same master nodes, perform the following steps:

1. Ensure that the replication agent and the cache agent are stopped and that applications are disconnected from both databases. This example uses the `basicDSN` database.

   ```
   ttCWAdmin -stop -dsn basicDSN
   ```

2. On the node where you want the new active database to reside, destroy the databases by using the `ttDestroy` utility.

   ```
   ttDestroy basicDSN
   ```

3. Restore the backup database.

   ```
   ttCWADmin -restore -dsn basicDSN
   ```

4. If there are cache groups in the database, drop and re-create the cache groups.
5. Re-create the active standby pair replication scheme.

   ```
   ttCWAdmin -create -dsn basicDSN
   ```

6. Start the active standby pair replication scheme.

   ```
   ttCWAdmin -start -dsn basicDSN
   ```

Manual recovery when RETURN TWOSAFE is configured

You can configure an active standby pair to have a return service of `RETURN TWOSAFE` by using the `ReturnServiceAttribute` Clusterware attribute in the `cluster.oracle.ini` file. When `RETURN TWOSAFE` is configured, the database logs may be available on one or both nodes after both nodes fail.

This `cluster.oracle.ini` example includes backup configuration in case the database logs are not available:

```
[basicTwosafeDSN]
MasterHosts=host1,host2
ReturnServiceAttribute=RETURN TWOSAFE
RepBackupDir=/shared_drive/dsbackup
RepBackupPeriod=3600
```

Perform these recovery tasks:

1. Ensure that the replication agent and the cache agent are stopped and that applications are disconnected from both databases.

   ```
   ttCWAdmin -stop -dsn basicTwosafeDSN
   ```

2. Drop the active standby pair.

   ```
   ttCWAdmin -drop -dsn basicTwosafeDSN
   ```

3. Decide whether the former active or standby database is more up to date and re-create the active standby pair using the chosen database. The command prompts you to choose the host on which the active database will reside.

   ```
   ttCWAdmin -create -dsn basicTwosafeDSN
   ```

   If neither database is usable, restore the database from backups.

   ```
   ttCWAdmin -restore -dsn basicTwosafeDSN
   ```

4. Start the active standby pair replication scheme.

   ```
   ttCWAdmin -start -dsn basicTwosafeDSN
   ```

When more than two master hosts fail

Approach a failure of more than two master hosts as a more extreme case of dual host failure. Use these guidelines:

- Address the root cause of the failure if it is something like a power outage or network failure.
- Identify or obtain at least two healthy hosts for the active and standby databases.
- Update the `MasterHosts` and `SubscriberHosts` entries in the `cluster.oracle.ini` file.
- See "Manual recovery for advanced availability" on page 7-22 and "Manual recovery for basic availability" on page 7-23 for guidelines on subsequent actions to take.

Performing a forced switchover after failure of the active database or host

If you want to force a switchover to the standby database without waiting for automatic recovery to be performed by TimesTen and Oracle Clusterware, you can write an application that uses Oracle Clusterware commands. These are the tasks to perform:

1. Use the `crs_stop` command to stop the `ttCRSmaster` resource on the active database. This causes the role of the standby database to change to active.
2. Use the `crs_start` command to restart the `ttCRSmaster` resource on the former active database. This causes the database to recover and become the standby database.

See *Oracle Clusterware Administration and Deployment Guide* for more information about the `crs_stop` and `crs_start` commands.

Planned maintenance

This section includes the following topics:

Changing the schema

To include or exclude a table, see "Making DDL changes in an active standby pair" on page 6-1.

To include or exclude a cache group, see "Making schema changes to active standby pairs in a grid" on page 7-16.

To create PL/SQL procedures, sequences, materialized views and indexes on tables with data, perform these tasks:

1. Enable the addition of the object to the active standby pair.

   ```
   ttCWAdmin -beginAlterSchema advancedDSN
   ```

2. Create the object.
3. If the object is a sequence and you want to include it in the active standby pair replication scheme, alter the active standby pair.

   ```
   ALTER ACTIVE STANDBY PAIR INCLUDE samplesequence;
   ```

4. Duplicate the change to the standby database.

   ```
   ttCWAdmin -endAlterSchema advancedDSN
   ```

To add or drop a subscriber database or alter database attributes, perform the following tasks:

1. Stop the replication agents on the databases in the active standby pair. These commands use the `advancedCacheDSN` as an example.

   ```
   ttCWAdmin -stop -dsn advancedCacheDSN
   ```

2. Drop the active standby pair.

   ```
   ttCWAdmin -drop  -dsn advancedCacheDSN
   ```

3. Modify the schema as desired.
4. Re-create the active standby pair replication scheme.

   ```
   ttCWAdmin -create -dsn advancedCacheDSN
   ```

5. Start the active standby pair replication scheme.

   ```
   ttCWAdmin -start -dsn advancedCacheDSN
   ```

Performing a rolling upgrade of Oracle Clusterware software

See *Oracle Clusterware Administration and Deployment Guide*.

Upgrading TimesTen

See "Upgrading TimesTen when using Oracle Clusterware" in *Oracle TimesTen In-Memory Database Installation Guide*.

Adding a read-only subscriber to an active standby pair

To add a read-only subscriber to an active standby pair replication scheme managed by Oracle Clusterware, perform these steps:

1. Stop the replication agents on all databases. This example uses the `advancedSubscriberDSN`, which already has a subscriber and is configured for advanced availability.

   ```
   ttCWAdmin -stop -dsn advancedSubscriberDSN
   ```

2. Drop the active standby pair.

   ```
   ttCWAdmin -drop -dsn advancedSubscriberDSN
   ```

3. Modify the `cluster.oracle.ini` file.

   - Add the subscriber to the `SubscriberHosts` attribute.
   - If the cluster is configured for advanced availability, add a virtual IP address to the `SubscriberVIP` attribute.

   See "Configuring advanced availability" on page 7-5 for an example using these attributes.

4. Create the active standby pair replication scheme.

   ```
   ttCWAdmin -create -dsn advancedSubscriberDSN
   ```

5. Start the active standby pair replication scheme.

   ```
   ttCWAdmin -start -dsn advancedSubscriberDSN
   ```

Removing a read-only subscriber from an active standby pair

To remove a read-only subscriber from an active standby pair, perform these steps:

1. Stop the replication agents on all databases. This example uses the `advancedSubscriberDSN`, which has a subscriber and is configured for advanced availability.

   ```
   ttCWAdmin -stop -dsn advancedSubscriberDSN
   ```

2. Drop the active standby pair.

   ```
   ttCWAdmin -drop -dsn advancedSubscriberDSN
   ```

3. Modify the `cluster.oracle.ini` file.

   - Remove the subscriber from the `SubscriberHosts` attribute or remove the attribute altogether if there are no subscribers left in the active standby pair.
   - Remove a virtual IP from the `SubscriberVIP` attribute or remove the attribute altogether if there are no subscribers left in the active standby pair.

4. Create the active standby pair replication scheme.

   ```
   ttCWAdmin -create -dsn advancedSubscriberDSN
   ```

5. Start the active standby pair replication scheme.

   ```
   ttCWAdmin -start -dsn advancedSubscriberDSN
   ```

Adding an active standby pair to a cluster

To add an active standby pair (with or without subscribers) to a cluster that is already managing an active standby pair, perform these tasks:

1. Create and populate a database on the host where you intend the active database to reside initially. See "Create and populate a TimesTen database on one host" on page 7-12.
2. Modify the `cluster.oracle.ini` file. This example adds `advSub2DSN` to the `cluster.oracle.ini` file that already contains the configuration for `advancedSubscriberDSN`. The new active standby pair is on different hosts from the original active standby pair.

```
[advancedSubscriberDSN]
MasterHosts=host1,host2,host3
SubscriberHosts=host4, host5
MasterVIP=192.168.1.1, 192.168.1.2
SubscriberVIP=192.168.1.3
VIPInterface=eth0
VIPNetMask=255.255.255.0

[advSub2DSN]
MasterHosts=host6,host7,host8
SubscriberHosts=host9, host10
MasterVIP=192.168.1.4, 192.168.1.5
SubscriberVIP=192.168.1.6
VIPInterface=eth0
VIPNetMask=255.255.255.0
```

3. Create new virtual IP addresses. On UNIX, the user must be `root` to do this.

```
ttCWAdmin -createVIPs -dsn advSub2DSN
```

4. Create the new active standby pair replication scheme.

```
ttCWAdmin -create -dsn advSub2DSN
```

5. Start the new active standby pair replication scheme.

```
ttCWAdmin -start -dsn advSub2DSN
```

Adding a read-only subscriber not managed by Oracle Clusterware

You can add a read-only subscriber that is not managed by Oracle Clusterware to an existing active standby pair replication scheme that is managed by Oracle Clusterware. Using the `ttCWAdmin -beginAlterSchema` command enables you to add the subscriber without dropping and recreating the replication scheme. Oracle Clusterware does not manage the subscriber because it is not part of the configuration that was set up for Oracle Clusterware management.

Perform these steps:

1. Enter the `ttCWAdmin -beginAlterSchema` command to stop the replication agent on the active and standby databases.
2. Using `ttIsql` to connect to the active database, add the subscriber to the replication scheme by using an `ALTER ACTIVE STANDBY PAIR` statement.

```
ALTER ACTIVE STANDBY PAIR ADD SUBSCRIBER ROsubDSN ON host6;
```

3. Enter the `ttCWAdmin -endAlterSchema` command to duplicate the standby database, register the altered replication scheme and start replication.
4. Enter the `ttIsql repschemes` command to verify that the read-only subscriber has been added to the replication scheme.

5. Use the `ttRepStateGet` procedure to verify that the state of the standby database is `STANDBY`.
6. On the subscriber host, use `ttRepAdmin -duplicate` to duplicate the standby database to the read-only subscriber. See "Duplicating a database" on page 4-2.
7. Start the replication agent on the subscriber host.

Rebuilding a read-only subscriber not managed by Oracle Clusterware

You can destroy and rebuild a read-only subscriber that is not managed by Oracle Clusterware. Perform these tasks:

1. Stop the replication agent on the subscriber host.
2. Use the `ttDestroy` utility to destroy the subscriber database.
3. On the subscriber host, use `ttRepAdmin -duplicate` to duplicate the standby database to the read-only subscriber. See "Duplicating a database" on page 4-2.

Removing an active standby pair from a cluster

To remove an active standby pair (with or without subscribers) from a cluster, perform these tasks:

1. Stop the replication agents on all databases in the active standby pair. This example uses `advSub2DSN`, which was added in "Adding an active standby pair to a cluster" on page 7-27.

   ```
   ttCWAdmin -stop -dsn advSub2DSN
   ```

2. Drop the active standby replication scheme.

   ```
   ttCWAdmin -drop -dsn advSub2DSN
   ```

3. Drop the virtual IP addresses for the active standby pair.

   ```
   ttCWAdmin -dropVIPs -dsn advSub2DSN
   ```

4. Modify the `cluster.oracle.ini` file (optional). Remove the entries for `advSub2DSN`.
5. If you want to destroy the databases, log onto each host that was included in the configuration for this active standby pair and use the `ttDestroy` utility.

   ```
   ttDestroy advSub2DSN
   ```

   For more information about `ttDestroy`, see "ttDestroy" in *Oracle TimesTen In-Memory Database Reference*.

Adding a host to the cluster

Adding a host requires that the cluster be configured for advanced availability. The examples in this section use the `advancedSubscriberDSN`.

To add two spare master hosts to a cluster, enter a command similar to the following:

```
ttCWAdmin -addMasterHosts -hosts "host8,host9" -dsn advancedSubscriberDSN
```

To add a spare subscriber host to a cluster, enter a command similar to the following:

```
ttCWAdmin -addSubscriberHosts -hosts "subhost1" -dsn advancedSubscriberDSN
```

Removing a host from the cluster

Removing a host from the cluster requires that the cluster be configured for advanced availability. `MasterHosts` must list more than two hosts if one of the master hosts is to be removed. `SubscriberHosts` must list at least one more host than the number of subscriber databases if one of the subscriber hosts is to be removed.

The examples in this section use the `advancedSubscriberDSN`.

To remove two spare master host from the cluster, enter a command similar to the following:

```
ttCWAdmin -delMasterHosts "host8,host9" -dsn advancedSubscriberDSN
```

To remove a spare subscriber hosts from the cluster, enter a command similar to the following:

```
ttCWAdmin -delSubscriberHosts "subhost1" -dsn advancedSubscriberDSN
```

Reversing the roles of the master databases

After a failover, the active and standby databases are on different hosts than they were before the failover. You can use the `-switch` option of the `ttCWAdmin` utility to restore the original configuration.

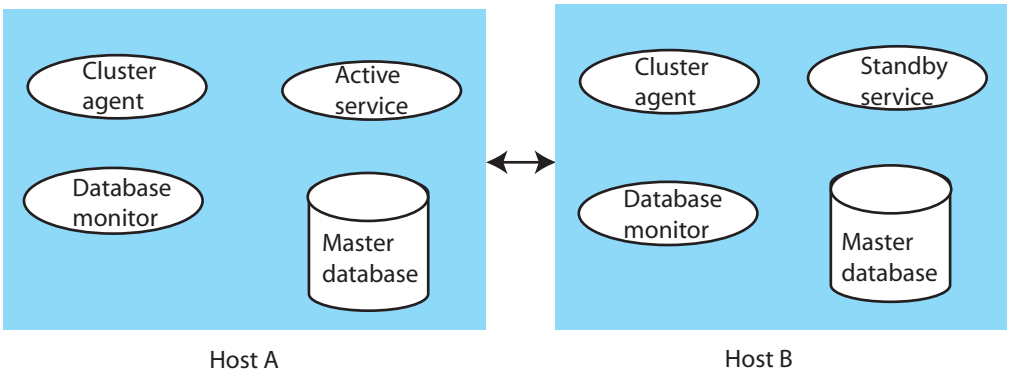
For example:

```
ttCWAdmin -switch -dsn basicDSN
```

Ensure that there are no open transactions before using the `-switch` option. If there are open transactions, the command fails.

Figure 7–6 shows the hosts for an active standby pair. The active database resides on host A, and the standby database resides on host B.

Figure 7–6 Hosts for an active standby pair



The `ttCWAdmin -switch` command performs these tasks:

- Deactivates the TimesTen cluster agent (`ttCRSAgent`) on host A (the active node)
- Disables the database monitor (`ttCRSmaster`) on host A
- Calls the `ttRepSubscriberWait`, `ttRepStop` and `ttRepDeactivate` built-in procedures on host A
- Stops the active service (`ttCRSActiveService`) on host A and reports a failure event to the Oracle Clusterware `CRSD` process

- Enables monitoring on host A and moves the active service to host B
- Starts the replication agent on host A, stops the standby service (`ttCRSsubservice`) on host B and reports a failure event to the Oracle Clusterware `CRSD` process on host B
- Starts the standby service (`ttCRSsubservice`) on host A

Moving a database to a different host

When a cluster is configured for advanced availability, you can use the `-relocate` option of the `ttCWAdmin` utility to move a database from the local host to the next available spare host specified in the `MasterHosts` attribute in the `cluster.oracle.ini` file. If the database on the local host has the active role, the `-relocate` option first reverses the roles. Thus the newly migrated active database becomes the standby and the standby becomes the active.

The `-relocate` option is useful for relocating a database if you decide to take the host offline. Ensure that there are no open transactions before you use the command.

For example:

```
ttCWAdmin -relocate -dsn advancedDSN
```

Performing host or network maintenance

If you decide to upgrade the operating system or hardware for a host or perform network maintenance, shut down Oracle Clusterware and disable automatic startup. Execute these Oracle Clusterware commands as `root` or OS administrator:

```
# crsctl stop crs

# crsctl disable crs
```

Shut down TimesTen. See "Shutting down a TimesTen application" in *Oracle TimesTen In-Memory Database Operations Guide*.

Perform the host maintenance. Then enable automatic startup and start Oracle Clusterware:

```
# crsctl enable crs

# crsctl start crs
```

See *Oracle Clusterware Administration and Deployment Guide* for more information about these commands.

Performing maintenance on the entire cluster

When all of the hosts in the cluster need to be brought down, stop Oracle Clusterware on each host individually. Execute these Oracle Clusterware commands as `root` or OS administrator:

```
# crsctl stop crs

# crsctl disable crs
```

Shut down TimesTen. See "Shutting down a TimesTen application" in *Oracle TimesTen In-Memory Database Operations Guide*.

Perform the maintenance. Then enable automatic startup and start Oracle Clusterware:

```
# crsctl enable crs

# crsctl start crs
```

See *Oracle Clusterware Administration and Deployment Guide* for more information about these commands.

Changing user names or passwords

When you create the active standby pair replication scheme with the `ttCWAdmin -create` command, Oracle Clusterware prompts for the user name and password of the internal user. If there are cache groups in the active standby pair, Oracle Clusterware also stores the cache administration user name and password. To change the user name or password for the internal user or the cache administration user, you must re-create the cluster.

To change the user name or password of the internal user that created the active standby pair replication or to change the cache administration user name or password, perform these tasks:

1. Stop the replication agents on the databases in the active standby pair. These commands use the `advancedCacheDSN` as an example.

   ```
   ttCWAdmin -stop -dsn advancedCacheDSN
   ```

2. Drop the active standby pair.

   ```
   ttCWAdmin -drop -dsn advancedCacheDSN
   ```

3. Change the appropriate user name or password:

   - Change the internal user name or password by using the `CREATE USER` or `ALTER USER` statements. See "Creating or identifying users to the database" in *Oracle TimesTen In-Memory Database Operations Guide*.
   - Change the cache administration user name or password by using the `ttCacheUidPwdSet` built-in procedure. See "Setting the cache administration user name and password" in *Oracle In-Memory Database Cache User's Guide*.

4. Re-create the active standby pair replication scheme.

   ```
   ttCWAdmin -create -dsn advancedCacheDSN
   ```

5. Start the active standby pair replication scheme.

   ```
   ttCWAdmin -start -dsn advancedCacheDSN
   ```

Monitoring cluster status

This section includes:

- Obtaining cluster status
- Message log files

Obtaining cluster status

Using the `-status` option of the `ttCWAdmin` utility reports information about all of the active standby pairs in an instance that are managed by the same instance administrator. If you specify the DSN, the utility reports information for the active standby pair with that DSN.

Example 7–1 Status after creating an active standby pair

After you have created an active standby pair replication scheme but have not yet started replication, `ttCWAdmin -status` returns information like this. Note that these grid states will be displayed before replication is started regardless of whether there is a cache grid.

```
$ ttCWAdmin -status
TimesTen Cluster status report as of Thu Nov 11 13:54:35 2010

==========================================================================
TimesTen daemon monitors:
Host:HOST1 Status: online
Host:HOST2 Status: online

==========================================================================
==========================================================================
TimesTen Cluster agents
Host:HOST1 Status: online
Host:HOST2 Status: online

==========================================================================

Status of Cluster related to DSN MYDSN:
==========================================================================
1. Status of Cluster monitoring components:
Monitor Process for Active datastore:NOT RUNNING
Monitor Process for Standby datastore:NOT RUNNING
Monitor Process for Master Datastore 1 on Host host1: NOT RUNNING
Monitor Process for Master Datastore 2 on Host host2: NOT RUNNING

2.Status of  Datastores comprising the cluster
Master Datastore 1:
Host:host1
Status:AVAILABLE
State:ACTIVE
Grid:NO GRID
Master Datastore 2:
Host:host2
Status:UNAVAILABLE
State:UNKNOWN
Grid:UNKNOWN
==========================================================================
The cluster containing the replicated DSN is offline
```

Example 7–2 Status when the active database is running

After you have started the replication scheme and the active database is running but the standby database is not yet running, `ttCWAdmin -status` returns information like this when a cache grid is not configured.

```
$ ttcwadmin -status
TimesTen Cluster status report as of Thu Nov 11 13:58:25 2010

==========================================================================
TimesTen daemon monitors:
Host:HOST1 Status: online
Host:HOST2 Status: online

==========================================================================
==========================================================================
```

```
TimesTen Cluster agents
Host:HOST1 Status: online
Host:HOST2 Status: online

========================================================================

Status of Cluster related to DSN MYDSN:
========================================================================
1. Status of Cluster monitoring components:
Monitor Process for Active datastore:RUNNING on Host host1
Monitor Process for Standby datastore:RUNNING on Host host1
Monitor Process for Master Datastore 1 on Host host1: RUNNING
Monitor Process for Master Datastore 2 on Host host2: RUNNING

2.Status of  Datastores comprising the cluster
Master Datastore 1:
Host:host1
Status:AVAILABLE
State:ACTIVE
Grid:NO GRID
Master Datastore 2:
Host:host2
Status:AVAILABLE
State:IDLE
Grid:NO GRID
========================================================================
The cluster containing the replicated DSN is online
```

If a cache grid is configured, then the last section appears as follows:

```
2.Status of  Datastores comprising the cluster
Master Datastore 1:
Host:host1
Status:AVAILABLE
State:ACTIVE
Grid:AVAILABLE
Master Datastore 2:
Host:host2
Status:AVAILABLE
State:IDLE
Grid:NO GRID
```

Example 7–3 Status when the active and the standby databases are running

After you have started the replication scheme and the active database and the standby database are both running, `ttCWAdmin -status` returns information like this when a cache grid is not configured.

```
$ ttcwadmin -status
TimesTen Cluster status report as of Thu Nov 11 13:59:20 2010

========================================================================
TimesTen daemon monitors:
Host:HOST1 Status: online
Host:HOST2 Status: online

========================================================================
========================================================================
TimesTen Cluster agents
Host:HOST1 Status: online
Host:HOST2 Status: online
```

```
========================================================================

Status of Cluster related to DSN MYDSN:
========================================================================
1. Status of Cluster monitoring components:
Monitor Process for Active datastore:RUNNING on Host host1
Monitor Process for Standby datastore:RUNNING on Host host2
Monitor Process for Master Datastore 1 on Host host1: RUNNING
Monitor Process for Master Datastore 2 on Host host2: RUNNING

2.Status of  Datastores comprising the cluster
Master Datastore 1:
Host:host1
Status:AVAILABLE
State:ACTIVE
Grid:NO GRID
Master Datastore 2:
Host:host2
Status:AVAILABLE
State:STANDBY
Grid:NO GRID
========================================================================
The cluster containing the replicated DSN is online
```

If a cache grid is configured, then the last section appears as follows:

```
2.Status of  Datastores comprising the cluster
Master Datastore 1:
Host:host1
Status:AVAILABLE
State:ACTIVE
Grid:AVAILABLE
Master Datastore 2:
Host:host2
Status:AVAILABLE
State:STANDBY
Grid:AVAILABLE
```

Message log files

The monitor processes report events and errors to the `ttcwerrors.log` and `ttcwmsg.log` files. The files are located in the *`daemon_home`*`/info` directory. The default size of these files is the same as the default maximum size of the user log. The maximum number of log files is the same as the default number of files for the user log. When the maximum number of files has been written, additional errors and messages overwrite the files, beginning with the oldest file.

For the default values for number of log files and log file size, see "Modifying informational messages" in *Oracle TimesTen In-Memory Database Operations Guide*.

8

TimesTen Configuration Attributes for Oracle Clusterware

The attributes defined in this chapter are used to set up TimesTen active standby pairs that are managed by Oracle Clusterware. These attributes are specified in the `cluster.oracle.ini` file. The `ttCWAdmin` utility creates and administers active standby pairs based on the information in the `cluster.oracle.ini` file.

List of attributes

This section lists the TimesTen configuration attributes for Oracle Clusterware in these tables:

- Table 8–1, " Required attributes"
- Table 8–2, " Conditionally required attributes"
- Table 8–3, " Optional attributes"

Table 8–1 Required attributes

Name	Description	Default
`MasterHosts`	Lists host names that may contain master databases in an active standby pair scheme.	None

Table 8–2 Conditionally required attributes

Name	Description	Default
`AppCheckCmd`	Command line for checking the status of a TimesTen application that is managed by Oracle Clusterware	None
`AppName`	The name of a TimesTen application that is managed by Oracle Clusterware	None
`AppStartCmd`	Command line for starting a TimesTen application that is managed by Oracle Clusterware	None
`AppStopCmd`	Command line for stopping a TimesTen application that is managed by Oracle Clusterware	None

Table 8–2 (Cont.) Conditionally required attributes

Name	Description	Default
AppType	The database to which the application should link.	None
CacheConnect	Specifies whether the active standby pair replicates cache groups.	N
GridPort	Lists the port numbers used by the cache grid agents for the active database and the standby database in an active standby pair that is a cache grid member.	None
MasterVIP	A list of two virtual IP addresses that can be associated with the master databases.	None
RemoteSubscriberHosts	A list of subscriber hosts that are not part of the cluster.	None
RepBackupDir	The directory to which the active database is backed up.	None
SubscriberHosts	List of host names that can contain subscriber databases.	None
SubscriberVIP	The list of virtual IP addresses that can be associated with subscriber databases.	None
VIPInterface	The name of the public network adapter that will be used for virtual IP addresses on each host.	None
VIPNetMask	The netmask of the virtual IP addresses.	None

Table 8–3 Optional attributes

Name	Description	Default
AppFailoverDelay	The number of seconds that the Oracle Clusterware resource that monitors the application waits after a failure is detected before performing a failover.	0
AppFailureInterval	The interval in seconds before which Oracle Clusterware stops a TimesTen application if the application has exceeded the number of failures specified by the Oracle Clusterware FAILURE_THRESHOLD resource attribute.	60

Table 8–3 (Cont.) Optional attributes

Name	Description	Default
AppFailureThreshold	The number of consecutive Oracle Clusterware resource failures that Oracle Clusterware tolerates for the action script for an application within an interval equal to 10 * AppScriptTimeout. The default is 2.	2
AppRestartAttempts	The number of times that Oracle Clusterware attempts to restart the TimesTen application on the current host before moving the application	100
AppScriptTimeout	The number of seconds the TimesTen application container waits for the action scripts to complete for a specific application.	60
AppUptimeThreshold	The number of seconds that a TimesTen application must be up before Oracle Clusterware considers the application to be stable.	600
AutoRecover	Specifies whether an active database should be automatically recovered from a backup if both master databases fail.	No
DatabaseFailoverDelay	The number of seconds that Oracle Clusterware waits before migrating a database to a new host after a failure.	60
FailureThreshold	The number of consecutive failures of resources managed by Oracle Clusterware that are tolerated within 10 seconds before the active standby pair is considered failed and a new active standby pair is created on spare hosts using the automated backup.	2
MasterStoreAttribute	A list of all desired replication scheme STORE attributes on master databases.	None
RepBackupPeriod	The number of seconds between each backup of the active database.	0 (disabled)
RepDDL	A SQL construct of the active standby pair scheme.	None
RepFullBackupCycle	The number times an incremental backup occurs between full backups.	5
ReturnServiceAttribute	The return service attribute of the active standby pair scheme.	None

Table 8–3 (Cont.) Optional attributes

Name	Description	Default
SubscriberStoreAttribute	The list of all desired replication scheme STORE attributes for the subscriber database.	None
TimesTenScriptTimeout	The number of seconds that Oracle Clusterware waits for the monitor process to start before assuming a failure.	1209600 seconds, or 14 days

Required attributes

These attributes must be present for each DSN in the `cluster.oracle.ini` file. They have no default values.

The required attributes are listed in Table 8–1, " Required attributes" and described in detail in this section.

MasterHosts

This attribute lists the host names that can contain master databases in the active standby pair. The first host listed has the active database when the cluster is started initially and after restarts. There are exceptions to the designated order:

- If there are already active and standby databases on specific nodes when the cluster is stopped, then the active and standby databases remain on those hosts when the cluster is restarted.
- If the cluster is started and the only existing database is on a host that is not listed first in `MasterHosts`, then that host will be configured with the active database. The first host listed for `MasterHosts` will be the standby database.

If the scheme contains no virtual IP addresses, only two master hosts are allowed.

Setting

Set `MasterHosts` as follows:

How the attribute is represented	Setting
`MasterHosts`	A comma-separated list of host names. The first host listed becomes the initial active database in the active standby pair.

Conditionally required attributes

These attributes may be required depending on the desired Oracle Clusterware configuration. They have no default values. The conditionally required attributes are listed in Table 8–2, " Conditionally required attributes" and described in detail in this section.

AppCheckCmd

This attribute specifies the full command line for executing a user-supplied script or program that checks the status of the TimesTen application specified by AppName. It must include the full path name of the executable. If there are spaces in the path name, enclose the path name in double quotes.

The command should be written to return 0 when the application is running and a nonzero number when the application is not running. When Oracle Clusterware detects a nonzero value, it takes action to recover the failed application.

Setting

Set AppCheckCmd as follows:

How the attribute is represented	Setting
AppCheckCmd	A string representing the command line for executing an application that checks the status of the application specified by AppName.

Examples

On UNIX:

```
AppCheckCmd=/mycluster/reader/app_check.sh check
```

On Windows:

```
AppCheckCmd="C:\Program Files\UserApps\UpdateApp.exe" -dsn myDSN -check
```

AppFailureInterval

This attribute sets the interval in seconds before which Oracle Clusterware stops a TimesTen application if the application has exceeded the number of failures specified by the Oracle Clusterware `FAILURE_THRESHOLD` resource attribute. If the value is zero, then failure tracking is disabled.

For more information about the Oracle Clusterware `FAILURE_THRESHOLD` resource attribute, see *Oracle Clusterware Administration and Deployment Guide*.

Setting

Set `AppFailureInterval` as follows:

How the attribute is represented	Setting
`AppFailureInterval`	The number of seconds in the interval before Oracle Clusterware stops an application. The default is 60. For example: `AppFailureInterval=120`

AppName

This attribute specifies the name of a TimesTen application managed by Oracle Clusterware. Oracle Clusterware uses the application name to name the corresponding resource. Any description of an application in the `cluster.oracle.ini` file must begin with this attribute.

Setting

Set `AppName` as follows:

How the attribute is represented	Setting
`AppName`	A string representing the name of the application. For example, `testApp`.

AppRestartAttempts

This attribute specifies the number of times that Oracle Clusterware attempts to restart the TimesTen application on the current host before moving the application to another host.

Setting

Set `AppRestartAttempts` as follows:

How the attribute is represented	Setting
`AppRestartAttempts`	The number of restart attempts. The default is 100. For example: `AppRestartAttempts=30`

AppStartCmd

This attribute specifies the command line that starts the TimesTen application specified by AppName. It must include the full path name of the executable. If there are spaces in the path name, enclose the path name in double quotes.

Setting

Set AppStartCmd as follows:

How the attribute is represented	Setting
AppStartCmd	A string that represents the command line for starting the application specified by AppName.

Examples

On UNIX:

```
AppCheckCmd=/mycluster/reader/app_start.sh start
```

On Windows:

```
AppCheckCmd="C:\Program Files\UserApps\UpdateApp.exe" -dsn myDSN -start
```

AppStopCmd

This attribute specifies the command line that stops the TimesTen application specified by AppName. It must include the full path name of the executable. If there are spaces in the path name, enclose the path name in double quotes.

Setting

Set AppStopCmd as follows:

How the attribute is represented	Setting
AppStopCmd	A string that represents the command line for stopping the application specified by AppName.

Examples

On UNIX:

```
AppCheckCmd=/mycluster/reader/app_stop.sh stop
```

On Windows:

```
AppCheckCmd="C:\Program Files\UserApps\UpdateApp.exe" -dsn myDSN -stop
```

AppType

This attribute determines the hosts on which the TimesTen application should start.

Setting

Set `AppType` as follows:

How the attribute is represented	Setting
`AppType`	`Active` - The application starts on the active database of an active standby pair. `Standby` - The application starts on the standby database of an active standby pair. If the standby database fails, applications linked to it migrate to the active database until a new standby database is available. `DualMaster` - The application starts on both the active host and the standby host. The failure of the application on the active host causes the active database and all other applications on the host to fail over to the standby host. `Subscriber` - The application starts on all subscriber databases. `Subscriber[`*`index`*`]` - The application starts on a subscriber database. The subscriber host used is the host occupying position *`index`* in either the `SubscriberHosts` attribute or the `SubscriberVIP` attribute, depending on whether virtual IP addresses are used. For a single subscriber, use `Subscriber[1]`. If no index is specified, TimesTen assumes that the application links to all subscribers.

AppUptimeThreshold

This attribute specifies the value for the Oracle Clusterware `UPTIME_THRESHOLD` resource attribute.The value represents the number of seconds that a TimesTen application must be up before Oracle Clusterware considers the application to be stable.

For more information about `UPTIME_THRESHOLD`, see *Oracle Clusterware Administration and Deployment Guide*.

How the attribute is represented	Setting
`AppUptimeThreshold`	Number of seconds. The default is 600. For example: `AppUptimeThreshold=60`

CacheConnect

If the active standby pair replicates cache groups, set this attribute to `Y`. If you specify `Y`, Oracle Clusterware assumes that TimesTen is connected to an Oracle database and prompts for the Oracle password.

Setting

Set `CacheConnect` as follows:

How the attribute is represented	Setting
`CacheConnect`	A value of `Y` (yes) or `N` (no). Default is `N`.

GridPort

This attribute lists the port numbers used by the cache grid agents for the active database and the standby database in an active standby pair that is a cache grid member. The port numbers are separated by a comma. This is a mandatory parameter when global cache groups are present.

Setting

Set `GridPort` as follows

How the attribute is represented	Setting
`GridPort`	Two port numbers separated by a comma. For example: `GridPort=16101,16102`

MasterVIP

This attribute is a list of the two virtual IP (VIP) addresses associated with two master databases. This is used for advanced availability. This attribute is required if you intend to use virtual IP addresses.

Setting

Set `MasterVIP` as follows:

How the attribute is represented	Setting
`MasterVIP`	A comma-separated list of two virtual IP addresses to the master databases.

RemoteSubscriberHosts

This attribute contains a list of subscriber hosts that are part of the active standby pair replication scheme but are not managed by Oracle Clusterware.

Setting

Set `RemoteSubscriberHosts` as follows:

How the attribute is represented	Setting
`RemoteSubscriberHosts`	A comma-separated list of subscriber hosts that are not managed by Oracle Clusterware.

RepBackupDir

This attribute indicates the directory where the backup of the active database is stored. This must be a directory in a shared file system that every node in the cluster can access. This attribute is required only if `RepBackupPeriod` is set to a value other than 0.

On UNIX, the directory must be a shared partition that is shared by all hosts in the cluster. On UNIX platforms, the partition must be NFS or OCFS (Oracle Cluster File System). On Windows, it must be an OCFS partition.

If you want to enable backup, install OCFS on the shared storage during the Oracle Clusterware installation process. You can use this shared storage for backup for an active standby pair.

See "Recovering from permanent failure of both master nodes" on page 7-8 and "Failure and recovery for active standby pair grid members" on page 7-16 for restrictions on backups.

Setting

Set `RepBackupDir` as follows:

How the attribute is represented	Setting
`RepbackupDir`	Full path name to the replication backup directory.

SubscriberHosts

Lists the host names that can contain subscriber databases. If virtual IP addresses are used, this list can overlap with the master host list provided by the `MasterHosts` attribute.

If the active standby pair is configured with subscribers, this attribute is required. It has no default value.

Setting

Set `SubscriberHosts` as follows:

How the attribute is represented	Setting
`SubscriberHosts`	A comma-separated list of host names. If virtual IP addresses are used, the order in which hosts are assigned to subscriber virtual IP addresses. If virtual IP addresses are not used, the order is used to determine which application with an `AppType` of `Subscriber[`*`index`*`]` is attached to the subscriber database on a specific host. Also, the number of subscriber hosts specified is the number of subscribers that are part of the active standby pair. A subscriber is brought up on every subscriber host.

SubscriberVIP

This attribute is a list of the virtual IP addresses associated with the subscriber databases. This is used for advanced availability. This attribute is required if you intend to use virtual IP addresses.

Setting

Set `SubscriberVIP` as follows:

How the attribute is represented	Setting
`SubscriberVIP`	One or more virtual IP addresses. These addresses are mapped to `SubscriberHosts`. The number of subscriber virtual IP addresses determines the number of subscribers that are brought up as part of the active standby pair. The order of subscriber virtual IP addresses is used to determine which application with an `AppType` of `Subscriber[`*`index`*`]` is attached to the database for a specific subscriber.

VIPInterface

This attribute is the name of the public network adapter used for virtual IP addresses on each host. This attribute is required if you intend to use virtual IP addresses.

Setting

Set `VIPInterface` as follows:

How the attribute is represented	Setting
`VIPInterface`	A string representing a network adapter.

VIPNetMask

This attribute is the netmask of the virtual IP addresses. This attribute is required if you intend to use virtual IP addresses.

Setting

Set `VIPNetMask` as follows:

How the attribute is represented	Setting
`VIPNetMask`	An IP netmask.

Optional attributes

These attributes are optional and have no default values. The optional attributes are listed in Table 8–3, " Optional attributes" and described in detail in this section.

AppFailoverDelay

This attribute specifies the number of seconds that the process that is monitoring the application waits after a failure is detected before performing a failover. The default is 0.

Setting

Set `AppFailoverDelay` as follows:

How the attribute is represented	Setting
`AppFailoverDelay`	An integer representing the number of seconds that the process that is monitoring the application waits after a failure is detected before performing a failover. The default is 0.

AppFailureThreshold

This attribute specifies the number of consecutive failures that Oracle Clusterware tolerates for the action script for an application within an interval equal to 10 * `AppScriptTimeout`. The default is 2.

Setting

Set `AppFailureThreshold` as follows:

How the attribute is represented	Setting
`AppFailureThreshold`	An integer indicating the number of consecutive failures that Oracle Clusterware tolerates for the action script for an application. The default is 2.

AppScriptTimeout

This attribute indicates the number of seconds that the TimesTen application monitor process waits for the start action script and the stop action script to complete for a specific application. The check action script has a nonconfigurable timeout of five seconds and is not affected by this attribute.

Setting

Set `AppScriptTimeout` as follows:

How the attribute is represented	Setting
`AppScriptTimeout`	An integer representing the number of seconds the TimesTen application container waits for start and stop action scripts to complete for a specific application. The default is 60.

AutoRecover

Specifies whether Oracle Clusterware automatically recovers the active database from the backup in the case of a failure of both masters.

If recovery is not automated (`AutoRecover=N`), the database can be recovered using the `ttCWAdmin -restore` command.

You cannot use `AutoRecover` if you are using cache groups in your configuration or if a cache grid is configured.

Setting

Set `AutoRecover` as follows:

How the attribute is represented	Setting
`AutoRecover`	`Y` - Oracle Clusterware automatically recovers the active database from the backup if both masters fail.
	`N` - In the case of the failure of both masters, you must recover manually. This is the default.

DatabaseFailoverDelay

This attributes specifies the number of seconds that Oracle Clusterware waits before migrating a database to a new host after a failure. Oracle Clusterware does not relocate a database if the database comes up during the delay period. This is applicable when advanced availability is configured. The default is 60 seconds.

Setting

Set `DatabaseFailoverDelay` as follows:

How the attribute is represented	Setting
`DatabaseFailoverDelay`	An integer representing the number of seconds that Oracle Clusterware waits before migrating a database to a new host after a failure. The default is 60.

FailureThreshold

This attribute specifies the number of consecutive failures of resources managed by Oracle Clusterware that are tolerated within 10 seconds before the active standby pair is considered failed and a new active standby pair is created on spare hosts using the automated backup. A spare node is only an option when using virtual IP addresses.

Oracle Clusterware tries to perform a duplicate for the active standby pair when a single failure occurs; it tries to perform a restoration if more than a single failure occurs.

This value is ignored for basic availability, since a spare node is only configured when at least one virtual IP address is configured, or is ignored when `RepBackupPeriod` is set to 0 when using advanced availability, which does include the configuration of at least one virtual IP address.

Note: TimesTen tolerates only one failure of a backup resource, regardless of the setting for this attribute.

Setting

Set `FailureThreshold` as follows:

How the attribute is represented	Setting
`FailureThreshold`	An integer representing the number of consecutive failures of resources managed by Oracle Clusterware that are tolerated within 10 seconds before the active standby pair is considered failed and a new active standby pair is created on spare hosts using the automated backup. The default is 2.

MasterStoreAttribute

This attribute indicates the desired replication scheme `STORE` attributes for the master databases. The `STORE` attributes apply to both the active and standby databases. The `STORE` clause for replication schemes is defined in *Oracle TimesTen In-Memory Database SQL Reference*.

This attribute is not required when `RepDDL` is configured.

If this attribute is not set, the `STORE` attributes take their default values. See "Setting STORE attributes" on page 3-6.

Setting

Set `MasterStoreAttribute` as follows:

How the attribute is represented	Setting
`MasterStoreAttribute`	The desired replication scheme `STORE` attributes for the master databases. For example: `PORT 20000 TIMEOUT 60`.

RepBackupPeriod

This attribute indicates the number of seconds between each backup of the active database. If this attribute is set to a value greater than 0, you must also specify a backup directory by setting `RepBackupDir`.

See "Recovering from permanent failure of both master nodes" on page 7-8 and "Failure and recovery for active standby pair grid members" on page 7-16 for restrictions on backups.

Setting

Set `RepBackupPeriod` as follows:

How the attribute is represented	Setting
`RepBackupPeriod`	An integer indicating the number of seconds between each backup of the active database. A value of 0 disables the backup process. The default is 0.

RepDDL

This attribute represents the SQL statement that creates the active standby pair. Use this attribute only in special circumstances. For example, you must specify RepDDL if you need to exclude tables and sequences from the active standby pair.

If RepDDL is set, do not set these attributes:

- ReturnServiceAttribute
- MasterStoreAttribute
- SubscriberStoreAttribute

Replace the database file name prefix in the SQL statement with the <DSN> macro. Use the <MASTERHOST[1]>, <MASTERHOST[2]> and <SUBSCRIBERHOST[*n*]> macros instead of the host names.

There is no default value for RepDDL.

This example sets RepDDL for two master databases:

```
RepDDL=CREATE ACTIVE STANDBY PAIR <DSN> ON <MASTERHOST[1]>, <DSN> ON
<MASTERHOST[2]>
```

See "Using the RepDDL attribute" on page 7-9 for additional examples.

You do not usually need to set the ROUTE clause in RepDDL because the transmitter of the replication agent automatically obtains the private and public network interfaces that Oracle Clusterware uses. However, if hosts have network connectivity for replication schemes that are not managed by Oracle Clusterware, then RepDDL needs to include the ROUTE clause.

If this attribute is used, each STORE clause must be followed by the pseudo host names such as:

- ActiveHost
- ActiveVIP
- StandbyHost
- StandbyVIP
- SubscriberHost
- SubscriberVIP

Setting

Set RepDDL as follows:

How the attribute is represented	Setting
RepDDL	Creates an active standby pair by issuing a CREATE ACTIVE STANDBY PAIR statement. There is no default value.

RepFullBackupCycle

This attribute specifies the number of incremental backups between full backups. The number of incremental backups depends on the capacity of the shared storage.

Setting this attribute can impact performance. There is a trade-off between the storage capacity and the time consumption for backup. An incremental backup can be performed much faster than a full backup. However, storage consumption increases until a full backup is performed.

See "Recovering from permanent failure of both master nodes" on page 7-8 and "Failure and recovery for active standby pair grid members" on page 7-16 for restrictions on backups.

Setting

Set `RepFullBackupCycle` as follows:

How the attribute is represented	Setting
`RepFullBackupCycle`	An integer value representing the number of incremental backups to perform between full backups. The default is 5.

ReturnServiceAttribute

This attribute specifies the return service for the active standby replication scheme. See "Using a return service" on page 3-4.

If no value is specified for this attribute, the active standby pair is configured with no return service.

Setting

Set `ReturnServiceAttribute` as follows:

How the attribute is represented	Setting
`ReturnServiceAttribute`	The type of return service. For example: `RETURN RECEIPT`. There is no default value.

SubscriberStoreAttribute

This attribute indicates the replication scheme `STORE` attributes of subscriber databases. The `STORE` attributes apply to all subscribers. The `STORE` clause for replication schemes is defined in *Oracle TimesTen In-Memory Database SQL Reference*.

This attribute is not required when `RepDDL` is present.

If this attribute is not set, the `STORE` attributes take their default values. See "Setting STORE attributes" on page 3-6.

Setting

Set `SubscriberStoreAttribute` as follows:

How the attribute is represented	Setting
`SubscriberStoreAttribute`	The list of `STORE` attributes and their values for the subscriber databases. For example: `PORT 20000 TIMEOUT 60`.

TimesTenScriptTimeout

This attribute specifies the number of seconds that Oracle Clusterware waits for the monitor process to start before assuming a failure.

Oracle TimesTen recommends setting a value of several hours because the action script may take a long time to duplicate the active database. The default is 1209600 seconds (14 days).

Setting

Set `TimesTenScriptTimeout` as follows:

How the attribute is represented	Setting
`TimesTenScriptTimeout`	An integer representing the number of seconds that Oracle Clusterware waits for the monitor process to start before assuming a failure. The default is 1209600 seconds (14 days).

9

Defining Replication Schemes

This chapter describes how to define replication schemes that are not active standby pairs. For information about defining active standby pair replication schemes, see Chapter 3, "Defining an Active Standby Pair Replication Scheme". If you want to replicate a database that has cache groups, see Chapter 5, "Administering an Active Standby Pair with Cache Groups".

To reduce the amount of bandwidth required for replication, see "Compressing replicated traffic" on page 9-23.

To replicate tables with columns in a different order or with a different number of partitions, see "Replicating tables with different definitions" on page 9-25.

This chapter includes these topics:

Designing a highly available system

These are the primary objectives of any replication scheme:

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

In a highly available system, a subscriber database must be able to survive failures that may affect the master. At a minimum, the master and subscriber need to be on separate hosts. 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.

In this chapter, we consider the replication schemes described in "Types of replication schemes" on page 1-6:

- Unidirectional
- Bidirectional split workload
- Bidirectional distributed workload
- Propagation

In addition, consider whether you want to replicate a whole database or selected elements of the database. Also consider the number of subscribers in the replication scheme. Unidirectional and propagation replication schemes allow you to choose the number of subscribers.

The rest of this section includes these topics:

- Considering failover and recovery scenarios
- Making decisions about performance and recovery tradeoffs
- Distributing workloads

For more information about using replication to facilitate online upgrades, see "Performing an online upgrade with replication" and "Performing an online upgrade with active standby pair replication" in *Oracle TimesTen In-Memory Database Installation Guide*.

Considering failover and recovery scenarios

As you plan a replication scheme, consider every failover and recovery scenario. For example, subscriber failures generally have no impact on the applications connected to the master databases. Their recovery does not disrupt user service. If a failure occurs on a master database, 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 database to one of its subscribers, and manage recovery of the failed database. See Chapter 11, "Managing Database Failover and Recovery".

When planning failover strategies, consider which subscribers will take on the role of the master and for which users or applications. Also consider recovery factors. For example, a failed master must be able to recover its database from its most up-to-date subscriber, and any subscriber must be able to recover from its master. A bidirectional scheme that replicates the entire database can take advantage of automatic restoration of a failed master. See "Automatic catch-up of a failed master database" on page 11-3.

Consider the failure scenario for the unidirectionally replicated database shown in Figure 9–1. In the case of a master failure, the application cannot access the database until it is recovered from the subscriber. You cannot switch the application connection or user load to the subscriber unless you use an `ALTER REPLICATION` statement to redefine the subscriber database as the master. See "Replacing a master database" on page 13-6.

Figure 9–1 Recovering a master in a unidirectional scheme

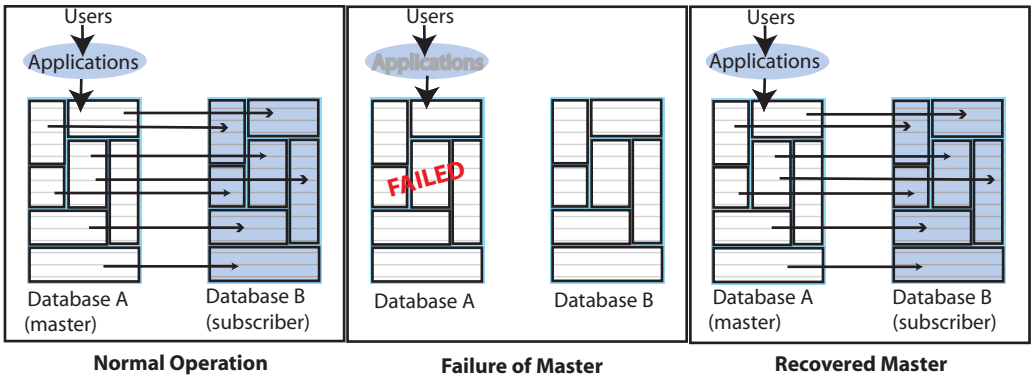
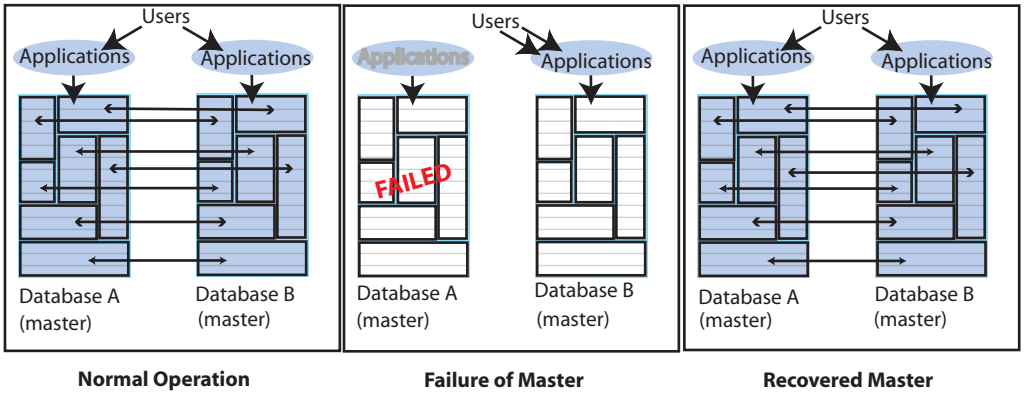


Figure 9–2 shows a bidirectional distributed workload scheme in which the entire database is replicated. Failover in this type of replication scheme involves shifting the users of the application on the failed database to the application on the surviving database. Upon recovery, the workload can be redistributed to the application on the recovered database.

Figure 9–2 Recovering a master in a distributed workload scheme



Similarly, the users in a split workload scheme must be shifted from the failed database to the surviving database. Because replication in a split workload scheme is not at the database level, you must use an `ALTER REPLICATION` statement to set a new master database. See "Replacing a master database" on page 13-6. Upon recovery, the users can be moved back to the recovered master database.

Propagation replication schemes also require the use of the `ALTER REPLICATION` statement to set a new master or a new propagator if the master or propagator fails. Higher availability is achieved if two propagators are defined in the replication scheme. See Figure 1–11 for an example of a propagation replication scheme with two propagators.

Making decisions about performance and recovery tradeoffs

When you design a replication scheme, 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 the 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.

You can configure replication to work asynchronously (the default), "semi-synchronously" with return receipt service, or fully synchronously with return twosafe service. Selecting a return service provides greater confidence that your data is consistent on the master and subscriber databases. Your decision to use default asynchronous replication or to configure return receipt or return twosafe mode depends on the degree of confidence you require and the performance tradeoff you are willing to make in exchange.

Table 9–1 summarizes the performance and recover tradeoffs of asynchronous replication, return receipt service and return twosafe service.

Table 9–1 Performance and recovery tradeoffs

Type of behavior	Asynchronous replication (default)	Return receipt	Return twosafe
Commit sequence	Each transaction is committed first on the master database.	Each transaction is committed first on the master database	Each transaction is committed first on the subscriber database.
Performance on master	Shortest response time and best throughput because there is no log wait between transactions or before the commit on the master.	Longer response time and less throughput than asynchronous. The application is blocked for the duration of the network round-trip after commit. Replicated transactions are more serialized than with asynchronous replication, which results in less throughput.	Longest response time and least throughput. The application is blocked for the duration of the network round-trip and remote commit on the subscriber before the commit on the master. Transactions are fully serialized, which results in the least throughput.
Effect of a runtime error	Because the transaction is first committed on the master database, errors that occur when committing on a subscriber require the subscriber to be either manually corrected or destroyed and then recovered from the master database.	Because the transaction is first committed on the master database, errors that occur when committing on a subscriber require the subscriber to be either manually corrected or destroyed and then recovered from the master database.	Because the transaction is first committed on the subscriber database, errors that occur when committing on the master require the master to be either manually corrected or destroyed and then recovered from the subscriber database.
Failover after failure of master	If the master fails and the subscriber takes over, the subscriber may be behind the master and must reprocess data feeds and be able to remove duplicates.	If the master fails and the subscriber takes over, the subscriber may be behind the master and must reprocess data feeds and be able to remove duplicates.	If the master fails and the subscriber takes over, the subscriber is at least up to date with the master. It is also possible for the subscriber to be ahead of the master if the master fails before committing a transaction it had replicated to the subscriber.

In addition to the performance and recovery tradeoffs 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 configuration or an active standby pair.

- Return twosafe allows you to specify a "local action" to be taken on the master database in the event of a timeout or other error encountered when replicating a transaction to the subscriber database.

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.

For more information about return services, see "Using a return service" on page 9-12.

Distributing workloads

Consider configuring the databases to distribute application workloads and make the best use of a limited number of servers. For example, it may be efficient and economical to configure the databases in a bidirectional distributed workload replication scheme so that each serves as both master and subscriber, rather than as separate master and subscriber databases. However, a distributed workload scheme works best with applications that primarily read from the databases. Implementing a distributed workload scheme for applications that frequently write to the same elements in a database may diminish performance and require that you implement a solution to prevent or manage update conflicts, as described in Chapter 14, "Resolving Replication Conflicts".

Defining a replication scheme

After you have designed your replication scheme, use the `CREATE REPLICATION` SQL statement to apply the scheme to your databases. You must have the `ADMIN` privilege to use the `CREATE REPLICATION` statement.

Table 9–2 shows the components of a replication scheme and identifies the clauses associated with the topics in this chapter. The complete syntax for the `CREATE REPLICATION` statement is provided in *Oracle TimesTen In-Memory Database SQL Reference*.

Table 9–2 Components of a replication scheme

Component	See...
`CREATE REPLICATION` *`Owner.SchemeName`*	"Owner of the replication scheme and replicated objects" on page 9-6
`ELEMENT` *`ElementName ElementType`*	"Defining replication elements" on page 9-7
`[`*`CheckConflicts`*`]`	"Checking for replication conflicts on table elements" on page 9-11
`{MASTER|PROPAGATOR}` *`DatabaseName`* `ON` *`"HostName"`*	"Database names" on page 9-6
`[TRANSMIT {NONDURABLE|DURABLE}]`	"Setting transmit durability on data store elements" on page 9-11
`SUBSCRIBER` *`DatabaseName`* `ON` *`"HostName"`*	"Database names" on page 9-6
`[`*`ReturnServiceAttribute`*`]`	"Using a return service" on page 9-12
`INCLUDE|EXCLUDE`	"Defining the DATASTORE element" on page 9-8

Table 9–2 (Cont.) Components of a replication scheme

Component	See...
STORE *DatabaseName DataStoreAttributes*	"Setting STORE attributes" on page 9-16
[*NetworkOperation*]	"Configuring network operations" on page 9-26

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

The replication scheme used by a database persists across system reboots. Modify a replication scheme by using the `ALTER REPLICATION` statement. See Chapter 13, "Altering Replication".

Owner of the replication scheme and replicated objects

The replication scheme and the replicated objects must be owned by the same user on every database in a replication scheme. To ensure that there is a common owner across all databases, you should explicitly specify the user and replication scheme in the `CREATE REPLICATION` statement.

For example, create a replication scheme named `repscheme` owned by user `repl`. The first line of the `CREATE REPLICATION` statement for `repscheme` is:

```
CREATE REPLICATION rep1.repscheme
```

Database names

These are the roles of the databases in a replication scheme:

- *Master*: Applications update the master database. The master sends the updates to the propagator or to the subscribers directly.
- *Propagator*: The propagator database receives updates from the master database and sends them to subscriber databases.
- *Subscriber*: Subscribers receive updates from the propagator or the master.

Before you define the replication scheme, you need to define the data source names (DSNs) for the databases in the replication scheme. On UNIX platforms, create an `odbc.ini` file. On Windows, use the ODBC Administrator to name the databases and set connection attributes. See "Step 1: Create the DSNs for the master and the subscriber" on page 2-5 for an example.

Each database "name" specified in a replication scheme must match the prefix of the database file name without the path specified for the `DataStore` data store attribute in the DSN definition. Use the same name for both the `DataStore` and `Data Source Name` data store attributes in each DSN definition. If the database path is *`directory/subdirectory/`*`foo.ds0`, then `foo` is the database name that you should use. For example, this entry in an `odbc.ini` file shows a `Data Source Name` (DSN) of `masterds`, while the `DataStore` value shows the path for `masterds`:

```
[masterds]
DataStore=/tmp/masterds
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

Table requirements and restrictions for replication schemes

The name and owner of replicated tables participating in the replication scheme must be identical on the master and subscriber databases. The column definitions of replicated tables participating in the replication scheme must be identical on the master and subscriber databases unless you specify a `TABLE DEFINITION CHECKING` value of `RELAXED` in the `CREATE REPLICATION` statement. If you specify `RELAXED`, then the tables must have the same key definition, number of columns and column data types. See "Replicating tables with different definitions" on page 9-25.

Replicated tables 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 database must also exist on its counterpart table in the subscriber.

> **Note:** The keys on replicated tables are transmitted in each update record to the subscribers. Smaller keys are transmitted more efficiently.

Replicated tables have these data type restrictions:

- `VARCHAR2`, `NVARCHAR2`, `VARBINARY` and `TT_VARCHAR` columns in replicated tables is limited to a size of 4 megabytes. 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 database character set. For example, if the character set requires four bytes for each character, the maximum possible length is one million characters. For an `NVARCHAR2` column, which requires two bytes for each character, the maximum length when using character length semantics is two million characters.
- Columns with the `BLOB` data type in replicated tables are limited to a size of 16 megabytes. Columns with the `CLOB` or `NCLOB` data type in replicated tables are limited to a size of 4 megabytes.
- A primary key column cannot have a LOB data type.

You cannot replicate tables with compressed columns.

If these requirements and restrictions present difficulties, you may want to consider using the Transaction Log API (XLA) as a replication mechanism. See "Using XLA as a replication mechanism" in *Oracle TimesTen In-Memory Database C Developer's Guide*.

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 database on which it is updated, and the subscriber databases to which the updates are replicated.

If you want to replicate a database with cache groups, see Chapter 5, "Administering an Active Standby Pair with Cache Groups".

These are restrictions on elements:

- Do not include a specific object (table, sequence or database) in more than one element description.
- Do not define the same element in the role of both master and propagator.
- An element must include the database on the current host as either the master, subscriber or propagator.
- Element names must be unique within a replication scheme.

The correct way to define elements in a multiple subscriber scheme is described in "Multiple subscriber schemes with return services and a log failure threshold" on page 9-28. The correct way to propagate elements is described in "Propagation scheme" on page 9-30.

The name of each element in a scheme can be used to identify the element if you decide later to drop or modify the element by using the `ALTER REPLICATION` statement.

You can add tables, sequences and databases to an existing replication scheme. See "Altering a replication scheme" on page 13-1. You can drop a table or sequence from a database that is part of a replication scheme after you exclude the table or sequence from the replication scheme. See "Dropping a table or sequence from a replication scheme" on page 13-4.

The rest of this section includes the following topics:

Defining the DATASTORE element

To replicate the entire contents of the master database (`masterds`) to the subscriber database (`subscriberds`), the `ELEMENT` description (named `ds1`) might look like the following:

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

Identify a database host using the host name returned by the `hostname` operating system command. It is good practice to surround a host name with double quotes.

You cannot replicate a temporary database.

You can choose to exclude certain tables and sequences from the `DATASTORE` element by using the `EXCLUDE TABLE` and `EXCLUDE SEQUENCE` clauses of the `CREATE REPLICATION` statement. When you use the `EXCLUDE` clauses, the entire database is replicated to all subscribers in the element *except* for the objects that are specified in the `EXCLUDE` clauses. Use only one `EXCLUDE TABLE` and one `EXCLUDE SEQUENCE` clause in an element description. For example, this element description excludes two tables and one sequence:

```
ELEMENT ds1 DATASTORE
  MASTER masterds ON "system1"
```

```
  SUBSCRIBER subscriberds ON "system2"
  EXCLUDE TABLE ttuser.tab1, ttuser.tab2
  EXCLUDE SEQUENCE ttuser.seq1
```

You can choose to include only certain tables and sequences in the database by using the INCLUDE TABLE and INCLUDE SEQUENCE 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 TABLE and one INCLUDE SEQUENCE clause in an element description. For example, this element description includes one table and two sequences:

```
ELEMENT ds1 DATASTORE
  MASTER masterds ON "system1"
  SUBSCRIBER subscriberds ON "system2"
  INCLUDE TABLE ttuser.tab3
  INCLUDE SEQUENCE ttuser.seq2, ttuser.seq3
```

Defining table elements

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

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

For requirements for tables in replication schemes, see "Table requirements and restrictions for replication schemes" on page 9-7.

Replicating tables with foreign key relationships

You may choose to replicate all or 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.

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 drop the replication scheme, create the new table with the foreign key relationship, and then create a new replication scheme replicating all of the related tables.

Replicating sequences

Sequences are replicated unless you exclude them from the replication scheme or unless they have the CYCLE attribute. Replication of sequences is optimized by reserving a range of sequence numbers on the standby database each time a sequence is updated on the active database. Reserving a range of sequence numbers reduces the number of updates to the transaction log. The range of sequence numbers is called a

cache. Sequence updates on the active database are replicated only when they are followed by or used in replicated transactions.

Consider a sequence `my.seq` with a `MINVALUE` of 1, an `INCREMENT` of 1 and the default *`Cache`* of 20. The very first time that you use `my.seq.NEXTVAL`, the current value of the sequence on the master database is changed to 2, and a new current value of 21 (20+1) is replicated to the subscriber. The next 19 references to `my.seq.NEXTVAL` on the master database result in no new current value being replicated, because the current value of 21 on the subscriber database is still ahead of the current value on the master. On the twenty-first reference to `my.seq.NEXTVAL`, a new current value of 41 (21+20) is transmitted to the subscriber database because the subscriber's previous current value of 21 is now behind the value of 22 on the master.

Sequence replication has these restrictions:

- Sequences with the `CYCLE` attribute cannot be replicated.
- The definition of the replicated sequence on each peer database must be identical.
- No conflict checking is performed on sequences. If you make updates to sequences in both databases 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 *nonreplicated* sequence with different `MINVALUE` and `MAXVALUE` attributes on each database to avoid conflicts. For example, you may create sequence `my.seq` on database `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 can make updates to either database using the sequence `my.seq` with the guarantee that the sequence values never conflict. Be aware that if you are planning to use `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 operation.

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 transactions on replicated tables. A side effect of this behavior is that these sequence updates are not purged from the log until followed by transactions on replicated tables. This causes `ttRepSubscriberWait` and `ttRepAdmin -wait` to fail when only these sequence updates are present at the end of the log.

To replicate the `ttuser.seq` sequence from a master database (named `masterds` and located on a host named `system1`) to a subscriber database (named `subscriberds` on a host named `system2`), the element description (named `a`) might look like the following:

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

Views and materialized views in a replicated database

A materialized view is a summary of data selected from one or more TimesTen tables, called detail tables. Although 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 databases 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 database.

If you replicate an entire database containing a materialized or nonmaterialized 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 database, 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.

Checking for replication conflicts on table elements

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

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

See Chapter 14, "Resolving Replication Conflicts" for a complete discussion on replication conflicts and how to configure the CHECK CONFLICTS clause in the CREATE REPLICATION statement.

Setting transmit durability on data store elements

A master database configured for asynchronous or return receipt replication is durable by default. This means that log records are committed to disk when transactions are committed. The master database can be set to nondurable by including the TRANSMIT NONDURABLE clause in the element description.

Transaction records in the master database log buffer are, by default, flushed to disk before they are forwarded to subscribers. If the entire master database 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 clause in the element description. The TRANSMIT setting has no effect on the subscriber. The transaction records on the subscriber database are always flushed to disk.

Master databases configured for return twosafe replication are nondurable by default and cannot be made durable. Setting TRANSMIT DURABLE on a database that is configured for return twosafe replication has no effect on return twosafe transactions.

Example 9–1 Replicating the entire master database with TRANSMIT NONDURABLE

To replicate the entire contents of the master database (masterds) to the subscriber database (subscriberds) and to eliminate the flush-log-to-disk operation, your element description (named a) might look like the following:

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

In general, if a master database fails, you have to initiate the `ttRepAdmin -duplicate` operation described in "Recovering a failed database" on page 11-5 to recover the failed master from the subscriber database. This is always true for a master database configured with `TRANSMIT DURABLE`.

A database configured as `TRANSMIT NONDURABLE` is recovered automatically by the subscriber replication agent if it is configured in the specific type of bidirectional scheme described in "Automatic catch-up of a failed master database" on page 11-3. Otherwise, you must follow the procedures described in "Recovering nondurable databases" on page 11-7 to recover a failed nondurable database.

Using a return service

You can configure your replication scheme with a return service to ensure a higher level of confidence that replicated data is consistent on both the master and subscriber databases. This section describes how to configure and manage the return receipt and return twosafe services.

You can specify a return service for table elements and database elements for any subscriber defined in a `CREATE REPLICATION` or `ALTER REPLICATION` statement.

Example 9–2 shows separate `SUBSCRIBER` clauses that can define different return service attributes for *`SubDatabase1`* and *`SubDatabase2`*.

Example 9–2 Different return services for each subscriber

```
CREATE REPLICATION Owner.SchemeName
  ELEMENT ElementNameElementType
    MASTER DatabaseName ON "HostName"
    SUBSCRIBER SubDatabase1 ON "HostName" ReturnServiceAttribute1
    SUBSCRIBER SubDatabase2 ON "HostName" ReturnServiceAttribute2;
```

Alternatively, you can specify the same return service attribute for all of the subscribers defined in an element. Example 9–3 shows the use of a single `SUBSCRIBER` clause that defines the same return service attributes for both *`SubDatabase1`* and *`SubDatabase2`*.

Example 9–3 Same return service for all subscribers

```
CREATE REPLICATION Owner.SchemeName
  ELEMENT ElementNameElementType
    MASTER DatabaseName ON "HostName"
    SUBSCRIBER SubDatabase1 ON "HostName",
               SubDatabase2 ON "HostName"
               ReturnServiceAttribute;
```

These sections describe the return service attributes:

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

RETURN RECEIPT

TimesTen provides an optional return receipt service to loosely couple or synchronize your application with the replication mechanism.

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 the application commits a transaction for an element on the master database, the application remains blocked until the subscriber acknowledges receipt of the transaction update. If the master is replicating the element to multiple subscribers, the application remains blocked until all of the subscribers have acknowledged receipt of the transaction update.

For example replication schemes that use return receipt services, see Example 9–24 and Example 9–25.

Example 9–4 RETURN RECEIPT

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

```
ELEMENT e TABLE 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, the application receives a `tt_ErrRepReturnFailed` (8170) warning on its commit request. 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 12-15 for more information on the return service timeout period.

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 9-18 for details.

The return receipt service is disabled by default if replication is stopped. See "RETURN SERVICES {ON | OFF} WHEN REPLICATION STOPPED" on page 9-20 for details.

RETURN RECEIPT BY REQUEST

`RETURN RECEIPT` enables notification of receipt for all transactions. You can use `RETURN RECEIPT BY REQUEST` 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` built-in 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 9–5 RETURN RECEIPT BY REQUEST

To enable confirmation that specific transactions committed on the `tab` table in the master database (`masterds`) are received by the subscriber (`subscriberds`), the element description (`e`) might look like:

```
ELEMENT e TABLE tab
    MASTER masterds ON "system1"
```

```
    SUBSCRIBER subscriberds ON "system2"
      RETURN RECEIPT BY REQUEST
```

Before committing a transaction that requires receipt notification, call the `ttRepSyncSet` built-in procedure to request the return services and to set the timeout period to 45 seconds:

```
Command> CALL ttRepSyncSet(0x01, 45, NULL);
```

If any of the subscribers are unable to acknowledge receipt of the transaction update within a configurable timeout period, the application receives a `tt_ErrRepReturnFailed` (8170) warning on its commit request. See "Setting the return service timeout period" on page 9-18.

You can use the `ttRepSyncGet` built-in procedure to check if a return service is enabled and obtain the timeout value. For example:

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

RETURN TWOSAFE BY REQUEST

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

The `ALTER TABLE` statement cannot be used to alter a replicated table that is part of a `RETURN TWOSAFE BY REQUEST` transaction. If `DDLCommitBehavior`=0 (the default), the `ALTER TABLE` operation succeeds because a commit is performed before the `ALTER TABLE` operation, resulting in the `ALTER TABLE` operation executing in a new transaction which is not part of the `RETURN TWOSAFE BY REQUEST` transaction. If `DDLCommitBehavior`=1, the `ALTER TABLE` operation results in error 8051.

Example 9–6 RETURN TWOSAFE BY REQUEST

To enable confirmation that specific transactions committed on the master database (`databaseA`) are also committed by the subscriber (`databaseB`), the element description (`a`) might look like:

```
ELEMENT a DATASTORE
    MASTER databaseA ON "system1"
    SUBSCRIBER databaseB ON "system2"
      RETURN TWOSAFE BY REQUEST;
```

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

```
Command> CALL ttRepSyncSet(0x01, 45, 1);
```

In this example, if the subscriber is unable to acknowledge commit of the transaction within the timeout period, the application receives a `tt_ErrRepReturnFailed`

(8170) warning on its commit request. The application can then chose how to handle the timeout. See "Setting the return service timeout period" on page 9-18.

You can use the `ttRepSyncGet` built-in procedure to check if a return service is enabled and obtain the timeout value. For example:

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

RETURN TWOSAFE

The return twosafe service ensures that each replicated transaction is committed on the subscriber database before it is committed on the master database. If replication is unable to verify the transaction has been committed on the subscriber, it returns notification of the error. Upon receiving an error, the application can either take a unique action or fall back on preconfigured actions, depending on the type of failure.

The return twosafe service is intended to be used in replication schemes where two databases must stay synchronized. One database has an active role, while the other database has a standby role but must be ready to assume an active role at any moment. Use return twosafe with a bidirectional replication scheme with exactly two databases.

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

Example 9–7 RETURN TWOSAFE

To confirm all transactions committed on the master database (`databaseA`) are also committed by the subscriber (`databaseB`), the element description (`a`) might look like the following:

```
ELEMENT a DATASTORE
    MASTER databaseA ON "system1"
    SUBSCRIBER databaseB ON "system2"
      RETURN TWOSAFE
```

The entire `CREATE REPLICATION` statement that specifies both `databaseA` and `databaseB` in a bidirectional configuration with `RETURN TWOSAFE` might look like the following:

```
CREATE REPLICATION bidirect
ELEMENT a DATASTORE
    MASTER databaseA ON "system1"
    SUBSCRIBER databaseB ON "system2"
      RETURN TWOSAFE
ELEMENT b DATASTORE
    MASTER databaseB ON "system2"
    SUBSCRIBER databaseA ON "system1"
      RETURN TWOSAFE;
```

When replication is configured with `RETURN TWOSAFE`, you must disable autocommit mode

When the application commits a transaction on the master database, the application remains blocked until the subscriber acknowledges it has successfully committed the transaction. Initiating identical updates or deletes on both databases 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 9-18.

NO RETURN

Use the `NO RETURN` attribute to explicitly disable the return receipt or return twosafe service. `NO RETURN` is the default condition. This attribute is typically set in `ALTER REPLICATION` statements. See Example 13–13.

Setting STORE attributes

Table 9–3 lists the optional `STORE` parameters for the `CREATE REPLICATION` and `ALTER REPLICATION` statements.

Table 9–3 STORE attribute descriptions

STORE attribute	Description
`DISABLE RETURN {SUBSCRIBER|ALL} NumFailures`	Set the return service policy so that return service blocking is disabled after the number of timeouts specified by *NumFailures*. See "Establishing return service failure/recovery policies" on page 9-19.
`RETURN SERVICES {ON|OFF} WHEN REPLICATION STOPPED`	Set return services on or off when replication is disabled. See "Establishing return service failure/recovery policies" on page 9-19.
`RESUME RETURN Milliseconds`	If `DISABLE RETURN` has disabled return service blocking, this attribute sets the policy for re-enabling the return service. See "Establishing return service failure/recovery policies" on page 9-19.
`RETURN WAIT TIME Seconds`	Specifies the number of seconds to wait for return service acknowledgement. A value of 0 means that there is no waiting. The default value is 10 seconds. The application can override this timeout setting by using the `returnWait` parameter in the `ttRepSyncSet` built-in procedure. See "Setting the return service timeout period" on page 9-18.
`DURABLE COMMIT {ON|OFF}`	Overrides the `DurableCommits` general connection attribute setting. `DURABLE COMMIT ON` enables durable commits regardless of whether the replication agent is running or stopped. See "DURABLE COMMIT" on page 9-22.

Table 9–3 (Cont.) STORE attribute descriptions

STORE attribute	Description
`LOCAL COMMIT ACTION {NO ACTION|ACTON}`	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. This is the default. `COMMIT`- On timeout, the commit function attempts to perform a commit to end 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. See "LOCAL COMMIT ACTION" on page 9-23.
`COMPRESS TRAFFIC {ON|OFF}`	Compress replicated traffic to reduce the amount of network bandwidth used. See "Compressing replicated traffic" on page 9-23.
`PORT` *`PortNumber`*	Set the port number used by subscriber databases 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 "Port assignments" on page 9-24.
`TIMEOUT` *`Seconds`*	Set the maximum number of seconds that the replication agent waits for a response from its database.
`FAILTHRESHOLD`	Set the log failure threshold. See "Setting the log failure threshold" on page 9-24.
`CONFLICT REPORTING {SUSPEND|RESUME} AT` *`Value`*	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. See Chapter 14, "Resolving Replication Conflicts".
`TABLE DEFINITION CHECKING {EXACT|RELAXED}`	Specify the type of table definition checking: ▪ `EXACT` - The tables must be identical on master and subscriber. This is the default. ▪ `RELAXED` - The tables must have the same key definition, number of columns and column data types. See "Replicating tables with different definitions" on page 9-25.

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 databases. 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 9–24.

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 9-12, a timeout occurs if any of the subscribers are unable to send an acknowledgement back to the master within the time period specified by `RETURN WAIT TIME`.

The default return service timeout period is 10 seconds. You can specify a different return service timeout period by:

- Configuring `RETURN WAIT TIME` in the `CREATE REPLICATION` or `ALTER REPLICATION` statement. A `RETURN WAIT TIME` of `0` indicates no waiting.
- Calling the `ttRepSyncSet` procedure with a new `returnWait` parameter

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.

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 mean the transaction has not been 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 9-18.

Example 9–8 Setting the timeout period for both databases in bidirectional replication scheme

To set the timeout period to 30 seconds for both bidirectionally replicated databases, `databaseA` and `databaseB`, in the `bidirect` replication scheme, the `CREATE REPLICATION` statement might look like the following:

```
CREATE REPLICATION bidirect
ELEMENT a DATASTORE
    MASTER databaseA ON "system1"
    SUBSCRIBER databaseB ON "system2"
      RETURN TWOSAFE
ELEMENT b DATASTORE
    MASTER databaseB ON "system2"
    SUBSCRIBER databaseA ON "system1"
      RETURN TWOSAFE
STORE databaseA RETURN WAIT TIME 30
STORE databaseB RETURN WAIT TIME 30;
```

Example 9–9 Resetting the timeout period

Use the `ttRepSyncSet` built-in procedure to reset the timeout period to 45 seconds. To avoid resetting the `requestReturn` and `localAction` values, specify `NULL`:

```
Command> CALL ttRepSyncSet(NULL, 45, NULL);
```

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. A subscriber may be unable to acknowledge a transaction that makes use of a return service and may time out with respect to the master. If any of the subscribers are unable to acknowledge the transaction update

within the timeout period, the 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:

- Configuring the `RETURN WAIT TIME` attribute in the `STORE` clause of the `CREATE REPLICATION` or `ALTER REPLICATION` statement
- Calling `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 been 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 the 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 to disable return receipt blocking for a particular subscriber. If 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 time out.
- Calling the `ttDurableCommit` procedure to durably commit transactions on the master that you can no longer verify as being received by the subscriber.

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

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.

The following attributes in the `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
- LOCAL COMMIT ACTION

The policies set by these attributes are applicable for the life of the database 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 database is set to `stop` or `pause` with respect to the master database (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 9–10 RETURN SERVICES ON WHEN REPLICATION STOPPED

Configure the `CREATE REPLICATION` statement to replicate updates from the `masterds` database to the `subscriber1` database. The `CREATE REPLICATION` statement specifies the use of `RETURN RECEIPT` and `RETURN SERVICES ON WHEN REPLICATION STOPPED`.

```
CREATE REPLICATION myscheme
ELEMENT e TABLE 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 database keeps track of the number of return receipt or return twosafe transactions that have exceeded the timeout period set by RETURN WAIT TIME. If the number of timeouts exceeds the maximum value set by DISABLE RETURN, the 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 enabled only when the replication agent is running. If DISABLE RETURN is specified but RESUME RETURN is not specified, the return services remain off until the replication agent for the database has been restarted. 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.

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. If one of the other subscribers later reaches the timeout threshold, the replication agent disables return service blocking for that subscriber also.

Example 9–11 DISABLE RETURN SUBSCRIBER

Configure the CREATE REPLICATION statement to replicate updates from the masterds database to the databases, subscriber1 and subscriber2. The 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 myscheme
ELEMENT e TABLE 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 9-20.

For another example that set the DISABLE RETURN attribute, see Example 9–12.

RESUME RETURN When we say return service blocking is "disabled," we mean that the applications on the master database no longer block execution while waiting to receive acknowledgements from the subscribers that they received or committed the replicated updates. Note, however, that the master still listens for an acknowledgement of each batch of replicated updates from the subscribers.

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.

Example 9–12 RESUME RETURN

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 myscheme
ELEMENT e TABLE ttuser.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 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 general connection attribute on the master database and forces durable commits regardless of whether the replication agent is running or stopped.

DURABLE COMMIT is useful if you have only one subscriber. However, if you are replicating the same data to two subscribers and you disable return service blocking to one subscriber, then you achieve better performance if you rely on the other subscriber than you would if you enable durable commits.

Example 9–13 DURABLE COMMIT ON

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 myscheme
ELEMENT e TABLE 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;
```

LOCAL COMMIT ACTION When using the return 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 the `CREATE REPLICATION` statement
- Calling the `ttRepSyncSet` procedure with the `localAction` parameter

The possible actions upon receiving a timeout during replication of a twosafe transaction are:

- `COMMIT` - On timeout, the commit function attempts to perform a commit to end the transaction locally. No more operations are possible on the same transaction.
- `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. This is the default

If the call returns with an error, you can use the `ttRepXactStatus` procedure described in "Checking the status of return service transactions" on page 12-15 to check 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 roll back the transaction on the master.

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 database specified by the `STORE` parameter in your `CREATE REPLICATION` or `ALTER REPLICATION` statement. TimesTen does not compress traffic from other databases.

Although the compression algorithm is optimized for speed, enabling the `COMPRESS TRAFFIC` attribute affects replication throughput and latency.

Example 9–14 Compressing traffic from one database

To compress replicated traffic from database `dsn1` and leave the replicated traffic from `dsn2` uncompressed, the `CREATE REPLICATION` statement looks like:

```
CREATE REPLICATION repscheme
ELEMENT d1 DATASTORE
    MASTER dsn1 ON host1
```

```
    SUBSCRIBER dsn2 ON host2
ELEMENT d2 DATASTORE
    MASTER dsn2 ON host2
    SUBSCRIBER dsn1 ON host1
STORE dsn1 ON host1 COMPRESS TRAFFIC ON;
```

Example 9–15 Compressing traffic between both databases

To compress the replicated traffic between both the `dsn1` and `dsn2` databases, use:

```
CREATE REPLICATION scheme
ELEMENT d1 DATASTORE
    MASTER dsn1 ON host1
    SUBSCRIBER dsn2 ON host2
ELEMENT d2 DATASTORE
    MASTER dsn2 ON host2
    SUBSCRIBER dsn1 ON host1
STORE dsn1 ON host1 COMPRESS TRAFFIC ON
STORE dsn2 ON host2 COMPRESS TRAFFIC ON;
```

Port assignments

Static port assignments are recommended. If you do not assign a `PORT` attribute, the TimesTen daemon dynamically selects the port. When ports are assigned dynamically for the replication agents, then the ports of the TimesTen daemons have to match as well. Setting the `PORT` attribute for one replication scheme sets it for all replication schemes.

You must assign static ports if you want to do online upgrades.

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 9–16 Assigning static ports

```
CREATE REPLICATION repscheme
ELEMENT el1 TABLE ttuser.tab
    MASTER dsn1 ON host1
    SUBSCRIBER dsn2 ON host2
ELEMENT el2 TABLE ttuser.tab
    MASTER dsn2 ON host2
    SUBSCRIBER dsn1 ON host1
STORE dsn1 ON host1 PORT 16080
STORE dsn2 ON host2 PORT 16083;
```

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. Use the `FAILTHRESHOLD` attribute to set the log failure threshold. See Example 9–24.

The default threshold value is 0, which means "no limit." See "Setting connection attributes for logging" on page 10-10 for details about log failure threshold values.

If a master sets a subscriber database 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 database. 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 replication state has been compromised.

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

Applications can use the ODBC `SQLGetInfo` function to check if the database it is connected to has been set to the `failed` state, as described in "Subscriber failures" on page 11-2.

Replicating tables with different definitions

Use the `TABLE DEFINITION CHECKING RELAXED` attribute to enable replication of tables that are not identical. For example, if tables have columns in a different order or have a different number of partitions, you can replicate them using this clause. A table has multiple partitions if columns have been added after its initial creation.

> **Note:** See Example 9–18 and "Check partition counts for the tables" in *Oracle TimesTen In-Memory Database Troubleshooting Guide* for more information.

Setting the `TABLE DEFINITION CHECKING` attribute to `RELAXED` requires that replicated tables have the same key definition, number of columns and column data types. Table definition checking occurs on the subscriber side. Setting this attribute to `RELAXED` for both master and subscriber has the same effect as setting it for only the subscriber.

The `RELAXED` setting can result in slightly slower performance. The change in performance depends on the workload and the number of partitions and columns in the tables. You can set table definition checking to `RELAXED` temporarily while consolidating tables with multiple partitions and then reset it to `EXACT`. There is no performance loss for tables with identical structures.

Example 9–17 Replicating tables with columns in different positions

Create table `t1` in `dsn1` database:

```
CREATE TABLE ttuser.t1 (a INT PRIMARY KEY, b INT, c INT);
```

Create table `ttuser.t1` in `dsn2` database with the columns in a different order than the columns in `ttuser.t1` in `dsn1` database. Note that the column names and data types are the same in both tables and `a` is the primary key in both tables.

```
CREATE TABLE ttuser.t1 (c INT, a INT PRIMARY KEY, b INT);
```

Create replication scheme `ttuser.rep1`. Set `TABLE DEFINITION CHECKING` to `RELAXED` for the subscriber, `dsn2`.

```
CREATE REPLICATION ttuser.rep1
       ELEMENT e1 TABLE ttuser.t1
       MASTER dsn1
       SUBSCRIBER dsn2
       STORE dsn2 TABLE DEFINITION CHECKING RELAXED;
```

Start the replication agent for both databases. Insert a row into `ttuser.t1` on `dsn1`.

```
Command> INSERT INTO ttuser.t1 VALUES (4,5,6);
1 row inserted.
```

Verify the results on `ttuser.t1` on `dsn2`.

```
Command> SELECT * FROM ttuser.t1;
< 5, 6, 4 >
1 row found.
```

Example 9–18 Replicating tables with a different number of partitions

When you alter a table to add columns, it increases the number of partitions in the table, even if you subsequently drop the new columns. You can use the `RELAXED` setting for `TABLE DEFINITION CHECKING` to replicate tables that have different number of partitions.

Create table `ttuser.t3` on `dsn1` with two columns.

```
CREATE TABLE ttuser.t3 (a INT PRIMARY KEY, b INT);
```

Create table `ttuser.t3` on `dsn2` with one column that is the primary key.

```
CREATE TABLE ttuser.t3 (a INT PRIMARY KEY);
```

Add a column to the table on `dsn2`. This increases the number of partitions to two, while the table on `dsn1` has one partition.

```
ALTER TABLE ttuser.t3 ADD COLUMN b INT;
```

Create the replication scheme on both databases.

```
CREATE REPLICATION reppart
       ELEMENT e2 TABLE ttuser.t3
       MASTER dsn1
       SUBSCRIBER dsn2
       STORE dsn2 TABLE DEFINITION CHECKING RELAXED;
```

Start the replication agent for both databases. Insert a row into `ttuser.t3` on `dsn1`.

```
Command> INSERT INTO ttuser.t3 VALUES (1,2);
1 row inserted.
```

Verify the results in `ttuser.t3` on `dsn2`.

```
Command> SELECT * FROM ttuser.t3;
< 1, 2 >
1 row found.
```

Configuring network operations

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.

The syntax of the `ROUTE` clause is:

```
ROUTE MASTER FullDatabaseName SUBSCRIBER FullDatabaseName
  {{MASTERIP MasterHost | SUBSCRIBERIP SubscriberHost}
    PRIORITY Priority} [...]
```

In dual master replication schemes, each master database is a subscriber of the other master database. This means that the CREATE REPLICATION statement should include ROUTE clauses in multiples of two to specify a route in both directions.

Example 9–19 Configuring multiple network interfaces

If host host1 is configured with a second interface accessible by the host name host1fast, and host2 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 repscheme
ELEMENT e1 TABLE ttuser.tab
    MASTER dsn1 ON host1
    SUBSCRIBER dsn2 ON host2
ELEMENT e2 TABLE ttuser.tab
    MASTER dsn2 ON host2
    SUBSCRIBER dsn1 ON host1
ROUTE MASTER dsn1 ON host1 SUBSCRIBER dsn2 ON host2
    MASTERIP host1fast PRIORITY 1
    SUBSCRIBERIP "192.168.1.100" PRIORITY 1
ROUTE MASTER dsn2 ON host2 SUBSCRIBER dsn1 ON host1
    MASTERIP "192.168.1.100" PRIORITY 1
    SUBSCRIBERIP host1fast 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 9–20 Configuring network priority

If host host1 is configured with two network interfaces at IP addresses 192.168.1.100 and 192.168.1.101, and host host2 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 repscheme
ELEMENT e TABLE ttuser.tab
  MASTER dsn1 ON host1
  SUBSCRIBER dsn2 ON host2
ROUTE MASTER dsn1 ON host1 SUBSCRIBER dsn2 ON host2
  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;
```

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.

Replication scheme syntax examples

The examples in this section illustrate how to configure a variety of replication schemes. The replication schemes include:

Single subscriber schemes

The scheme shown in Example 9–21 is a single master and subscriber unidirectional replication scheme. The two databases are located on separate hosts, `system1` and `system2`. We use the `RETURN RECEIPT` service to confirm that all transactions committed on the `ttuser.tab` table in the master database are received by the subscriber.

Example 9–21 Replicating one table

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

The scheme shown in Example 9–22 is a single master and subscriber unidirectional replication scheme. The two databases are located on separate hosts, `server1` and `server2`. The master database, named `masterds`, replicates its entire contents to the subscriber database, named `subscriberds`.

Example 9–22 Replicating entire database

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

Multiple subscriber schemes with return services and a log failure threshold

You can create a replication scheme that includes up to 128 subscriber databases. If you are configuring propagator databases, you can configure up to 128 propagators. Each propagator can have up to 128 subscriber databases. See "Propagation scheme" on page 9-30 for an example of a replication scheme with propagator databases.

Example 9–23 Replicating to two subscribers

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

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

Example 9–24 Replicating to two subscribers with RETURN RECEIPT

This example uses the basic example in Example 9–23 and adds a RETURN RECEIPT attribute and a STORE parameter. RETURN RECEIPT enables the return receipt service for both databases. The STORE parameter sets a FAILTHRESHOLD value of 10 to establish the maximum number of transaction log files that can accumulate on masterds for a subscriber before it assumes the subscriber has failed.

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

Example 9–25 Enabling RETURN RECEIPT for only one subscriber

This example shows how to enable RETURN RECEIPT for only subscriber2ds. Note that there is no comma after the subscriber1ds definition.

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

Example 9–26 Enabling different return services for subscribers

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

Replicating tables to different subscribers

The replication scheme shown in Example 9–27 establishes a master database, named centralds, that replicates four tables. ttuser.tab1 and ttuser.tab2 are replicated to the subscriber backup1ds. ttuser.tab3 and ttuser.tab4 are replicated to backup2ds. The master database is located on the finance server. Both subscribers are located on the backupsystem server.

Example 9–27 Replicating tables to different subscribers

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

```
  MASTER centralds ON "finance"
  SUBSCRIBER backup2ds ON "backupsystem"
ELEMENT d TABLE ttuser.tab4
  MASTER centralds ON "finance"
  SUBSCRIBER backup2ds ON "backupsystem";
```

Propagation scheme

In Example 9–28, the master database sends updates on a table to a propagator that forwards the changes to two subscribers. The master database is `centralds` on the `finance` host. The propagator database is `propds` on the `nethandler` host. The subscribers are `backup1ds` on `backupsystem1` and `backup2ds` on `backupsystem2`.

The replication scheme has two elements. For element `a`, the changes to the `tab` table on `centralds` are replicated to the `propds` propagator database. For element `b`, the changes to the `tab` table received by `propds` are replicated to the two subscribers, `backup1ds` and `backup2ds`.

Example 9–28 Propagation

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

Bidirectional split workload schemes

In Example 9–29, there are two databases, `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` database updates the `waccounts` table and replicates it to the `eastds` database. The `eaccounts` table is owned by the `eastds` database and is replicated to the `westds` database. The `RETURN RECEIPT` attribute enables the return receipt service to guarantee that transactions on either master table are received by their subscriber.

Example 9–29 Bidirectional split workload

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

Bidirectional distributed workload scheme

Example 9–30 shows a bidirectional general workload replication scheme in which the `ttuser.accounts` table can be updated on either the `eastds` or `westds` database. Each database is both a master and a subscriber for the `accounts` table.

Note: Do not use a bidirectional distributed workload replication scheme with return twosafe return service.

Example 9–30 Bidirectional distributed workload scheme

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

When elements are replicated in this manner, the applications should write to each database in a coordinated manner to avoid simultaneous updates on the same data. To manage update conflicts, include a timestamp column of type `BINARY(8)` in the replicated table and enable timestamp comparison by including the `CHECK CONFLICTS` clause in the `CREATE REPLICATION` statement. See Chapter 14, "Resolving Replication Conflicts" for a complete discussion on how to manage update conflicts.

Example 9–31 shows that the `tstamp` timestamp column is included in the `ttuser.accounts` table. The `CREATE REPLICATION` statement has been modified to include the `CHECK CONFLICTS` clause.

Example 9–31 Managing update conflicts

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

CREATE REPLICATION r1
ELEMENT elem_accounts_1 TABLE ttuser.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 ttuser.accounts
  CHECK CONFLICTS BY ROW TIMESTAMP
    COLUMN tstamp
    UPDATE BY SYSTEM
    ON EXCEPTION ROLLBACK WORK
  MASTER eastds ON "eastcoast"
  SUBSCRIBER westds ON "westcoast";
```

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 9–32. Entering the element description for the five tables, `ttuser.accounts`, `ttuser.sales`,

ttuser.orders, ttuser.inventory, and ttuser.customers, would be tedious and error-prone if done manually.

Example 9–32 General workload bidirectional replication scheme

```
CREATE REPLICATION bigscheme
ELEMENT elem_accounts_1 TABLE ttuser.accounts
  MASTER westds ON "westcoast"
  SUBSCRIBER eastds ON "eastcoast"
ELEMENT elem_accounts_2 TABLE ttuser.accounts
  MASTER eastds ON "eastcoast"
  SUBSCRIBER westds ON "westcoast"
ELEMENT elem_sales_1 TABLE ttuser.sales
  MASTER westds ON "westcoast"
  SUBSCRIBER eastds ON "eastcoast"
ELEMENT elem_sales_2 TABLE ttuser.sales
  MASTER eastds ON "eastcoast"
  SUBSCRIBER westds ON "westcoast"
ELEMENT elem_orders_1 TABLE ttuser.orders
  MASTER westds ON "westcoast"
  SUBSCRIBER eastds ON "eastcoast"
ELEMENT elem_orders_2 TABLE ttuser.orders
  MASTER eastds ON "eastcoast"
  SUBSCRIBER westds ON "westcoast"
ELEMENT elem_inventory_1 TABLE ttuser.inventory
  MASTER westds ON "westcoast"
  SUBSCRIBER eastds ON "eastcoast"
ELEMENT elem_inventory_2 TABLE ttuser.inventory
  MASTER eastds ON "eastcoast"
  SUBSCRIBER westds ON "westcoast"
ELEMENT elem_customers_1 TABLE ttuser.customers
  MASTER westds ON "westcoast"
  SUBSCRIBER eastds ON "eastcoast"
ELEMENT elem_customers_2 TABLE ttuser.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 9–33 can be used to build the scheme shown in Example 9–32.

Example 9–33 Using a Perl script to create a replication scheme

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

print "CREATE REPLICATION 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 9–33 can be obtained from some other source, such as a database. For example, you can use `ttIsql` and `f` in a Perl statement to generate a `@tables` array for all of the tables in the `WestDSN` database with the owner name `repl`:

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

Example 9–34 shows a modified version of the script in Example 9–33 that creates a replication scheme for all of the `repl` tables in the `WestDSN` database. (Note that some substitution may be necessary to remove extra spaces and line feeds from the `grep` output.)

Example 9–34 Perl script to create a replication scheme for all tables in WestDSN

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

print "CREATE REPLICATION 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";
```

10

Setting Up a Replicated System

This chapter describes how to set up and start replication. All of the topics in this chapter apply to replication schemes that are not active standby pairs. Some of the topics in this chapter also apply to active standby pairs.

To set up an active standby pair, see:

- "Setting up an active standby pair with no cache groups" on page 4-3
- "Setting up an active standby pair with a read-only cache group" on page 5-2
- "Setting up an active standby pair with an AWT cache group" on page 5-4

This chapter includes the following topics:

- Configuring the network
- Setting up the replication environment
- Applying a replication scheme to a database
- Duplicating a master database to a subscriber
- Configuring a large number of subscribers
- Replicating databases across releases
- Starting and stopping the replication agents
- Setting the replication state of subscribers

Configuring the network

This section applies to both active standby pairs and other replication schemes. It describes some of the issues to consider when replicating TimesTen data over a network. The topics include:

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

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 databases.

Table 10–1 provides guidelines for calculating the size of replicated records.

Table 10–1 Replicated record sizes

Record Type	Size
Begin transaction	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

Transactions are sent between replicated databases in batches. A batch is created whenever there is no more data in the transaction log buffer in the master database, or when the current batch is roughly 256K bytes. See "Copying updates between databases" on page 1-2 for more information.

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 the `CREATE ACTIVE STANDBY PAIR` or `CREATE REPLICATION` statement. See "Compressing replicated traffic" on page 9-23 for details.

See installation information for your platform in *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 on which your database 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 for each host.

This section includes these topics:

- Identifying database hosts and network interfaces using the ROUTE clause
- Identifying database hosts on UNIX without using the ROUTE clause
- Host name resolution on Windows
- User-specified addresses for TimesTen daemons and subdaemons

Identifying database hosts and network interfaces using the ROUTE clause

When specifying the host for a database in a replication element, you should always use the name returned by the `hostname` command, as replication uses the 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 "Configuring network operations" on page 9-26 for more information.

Identifying database hosts on UNIX without using the ROUTE clause

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:

When possible, you should use the `ROUTE` clause of a replication scheme to identify database hosts and the network interfaces to use for replication. However, if you have a legacy replication configuration that does not use the `ROUTE` clause, this section explains how to configure operating system and DNS files for a replication host with multiple network interfaces.

If a host contains multiple network interfaces (with different IP addresses) and replication is not configured with a `ROUTE` clause, TimesTen replication tries to connect to the IP addresses in the same order as returned by the `gethostbyname` call. It will try to connect using the first address; if a connection cannot be established, it tries the remaining addresses in order until a connection is established. TimesTen replication uses this same sequence each time it establishes a new connection to a host. If a connection to a host fails on one IP address, TimesTen replication attempts to re-connect (or fall back) to another IP address for the host in the same manner described above.

There are two basic ways you can configure a host to use multiple IP addresses on UNIX platforms: DNS or the `/etc/hosts` file.

Note: If you have multiple network interface cards (NICs), be sure that "multi on" is specified in the `/etc/host.conf` file. Otherwise, `gethostbyname` will not return multiple addresses.

For example, if your machine has two NICs, use the following syntax for your `/etc/hosts` file:

```
127.0.0.1  localhost
IP_address_for_NIC_1  official_hostname optional_alias
IP_address_for_NIC_2  official_hostname optional_alias
```

The host name *official_hostname* is the name returned by the `hostname` command.

When editing the `/etc/hosts` file, keep in mind that:

- You must log in as `root` to change the `/etc/hosts` file.
- There should only be one line per IP address.
- There can be multiple alias names on each line.
- When there are multiple IP addresses for the same host name, they must be on consecutive lines.
- The host name can be up to 30 characters long.

For example, the following entry in the `/etc/hosts` file on a UNIX platform describes a server named `Host1` with two IP addresses:

```
127.0.0.1        localhost
10.10.98.102     Host1
192.168.1.102    Host1
```

To specify the same configuration for DNS, your entry in the domain zone file would look like:

```
Host1     IN     A     10.10.98.102
          IN     A     192.168.1.102
```

In either case, you only need to specify `Host1` as the host name in your replication scheme and replication will use the first available IP address when establishing a connection.

In an environment in which multiple IP addresses are used, you can also assign multiple host names to a single IP address in order to restrict a replication connection to a specific IP address. For example, you might have an entry in your `/etc/hosts` file that looks like:

```
127.0.0.1        localhost
10.10.98.102     Host1
192.168.1.102    Host1 RepHost1
```

or a DNS zone file that looks like:

```
Host1     IN     A     10.10.98.102
          IN     A     192.168.1.102
RepHost1  IN     A     192.168.1.102
```

If you want to restrict replication connections to IP address 192.168.1.102 for this host, you can specify `RepHost1` as the host name in your replication scheme. Another option is to simply specify the IP address as the host name in the `CREATE REPLICATION` statement used to configure your replication scheme.

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 options, a Windows machine resorts to broadcasting to detect peer nodes, which is extremely slow.

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 database host in a replication scheme. Do not identify a host using its IP address for one database and then use its host name for the same or another database.

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 database

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 `ttHostNameSet` built-in procedure may be used in this instance to explicitly indicate to TimesTen that the current database is in fact the database specified in the replication scheme. See "ttHostNameSet" in *Oracle TimesTen In-Memory Database Reference* for more information.

Setting up the replication environment

The topics related to setting up your replication environment include:

- Establishing the databases
- Connection attributes for replicated databases
- Configuring parallel replication
- Managing the transaction log on a replicated database

Establishing the databases

You can replicate one or more tables on any existing database. If the database you want to replicate does not yet exist, you must first create one, as described in "Managing TimesTen Databases" in *Oracle TimesTen In-Memory Database Operations Guide*.

After you have identified or created the master database, create a DSN definition for the subscriber database on the target host. Set the connection attributes for the master and subscriber databases as described in "Connection attributes for replicated databases" on page 10-6.

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

- Connect to the database and use SQL statements to create new tables in the subscriber database that match those to be replicated from the master.
- Use the `ttRepAdmin -duplicate` utility to copy the entire contents of the master database to the subscriber. See "Duplicating a master database to a subscriber" on page 10-12.

Connection attributes for replicated databases

Databases that replicate to each other must have the same `DatabaseCharacterSet` data store attribute. TimesTen does not perform any character set conversion between replicated databases.

If you wish to configure parallel replication, see "Configuring parallel replication" on page 10-7 for information about setting the `ReplicationParallelism` and `ReplicationApplyOrdering` data store attributes.

See "Setting connection attributes for logging" on page 10-10 for recommendations for managing the replication log files.

It is possible to replicate between databases with different settings for the `TypeMode` data store attribute. However, you must make sure that the underlying data type for each replicated column is the same on each node. See "TypeMode" in *Oracle TimesTen In-Memory Database Reference* for more information.

In an active standby pair, use the `ReceiverThreads` first connection attribute to increase the number of threads that apply changes from the active database to the standby database from 1 to 2. If you set `ReceiverThreads` to 2 on the standby, you should also set it to 2 on the active to maintain increased throughput if there is a failover.

You can also set `ReceiverThreads` to 2 on one or more read-only subscribers in an active standby pair to increase replication throughput from the standby database.

Databases must be hosted on systems that have two or more CPUs to take advantage of setting this attribute to 2.

Configuring parallel replication

By default, replication is performed with a single thread where the nodes in a replication scheme have one log reader, or transmitter thread, on the source database, and one applier, or receiving thread, on the target database. You can increase your performance by configuring parallel replication, which configures multiple threads for sending updates from the source database to the target database and for applying the updates on the target database.

> **Note:** If you enable parallel replication, you cannot execute both DDL and DML statements in the same transaction.

There are two types of parallel replication, each of which is configured with the `ReplicationApplyOrdering` and `ReplicationParallelism` data store creation attributes and must be set when the database is created. Since both `ReplicationParallelism` and `ReplicationApplyOrdering` attributes are data store attributes, they cannot be modified after database creation.

> **Note:** All databases within the replication scheme that use parallel replication must be configured identically with the same type of parallel replication and the same number of threads or tracks.
>
> The only time you can have different values for parallel replication attributes is during an upgrade. For details, see "Performing an upgrade on databases that use parallel replication" in the *Oracle TimesTen In-Memory Database Installation Guide*.

The following sections describe both options for parallel replication:

- Configuring automatic parallel replication
- Configuring user-defined parallel replication for other replication schemes

Configuring automatic parallel replication

Automatic parallel replication enables you to configure multiple threads that act in parallel to replicate and apply transactional changes to nodes in a replication scheme. Automatic parallel replication enforces transactional dependencies and applies changes in commit order.

Automatic parallel replication is enabled by default with `ReplicationApplyOrdering=0`. To configure parallel replication, set `ReplicationParallelism` to a number from 2 to 32. The number cannot exceed half the value of `LogBufParallelism`. This number indicates the number of transmitter threads on the source database and the number of receiver threads on the target database. The default for `ReplicationParallelism` is 1, which indicates single-threaded replication.

> **Note:** If `ReplicationParallelism` is greater than 1, the `LogBufParallelism` first connection attribute must be an integral multiple of `ReplicationParallelism`.

If the replication scheme is an active standby pair that replicates AWT cache groups, the settings for `ReplicationApplyOrdering`, `ReplicationParallelism` and

the `CacheAwtParallelism` data store attributes determine how many threads are used to apply changes in the TimesTen cache tables to the corresponding Oracle tables. See "Configuring parallel propagation to Oracle tables" in *Oracle In-Memory Database Cache User's Guide* for more information.

For more information on these data store attributes, see "ReplicationParallelism", "ReplicationApplyOrdering", and "LogBufParallelism" in the *Oracle TimesTen In-Memory Database Reference*.

Configuring user-defined parallel replication for other replication schemes

If your application has predictable transactional dependencies and does not require the commit order on the target database be the same as the order on the source database, you can increase replication throughput by using user-defined parallel replication, which enables the user to manually divide work across different tracks.

User-defined parallel replication configures multiple threads for sending updates from the source database to the target database and for applying the updates on the target database. The application assigns transactions to tracks. The application specifies which track a transaction belongs to when the transaction starts on the source database. The transactions in each track are applied in the order in which they are received on the target database, but commit order is not maintained for transactions across the different tracks.

> **Note:** Use caution in assigning tracks to transactions that affect tables with foreign key relationships. If transactions on related tables are assigned to different tracks, one of the transactions can be lost because the transactions may be applied out of commit order.

In general, transactions that modify the same table should be assigned to the same replication track. In addition, updates that should be applied in order on the receiving side should use the same track. However, if all transactions insert to a particular table, they can be assigned to different tracks to increase replication throughput. You can split the workload for a table across multiple tracks with a key that ties a row to the same track.

Enable user-defined parallel replication by setting these data store attributes at database creation time:

- Set `ReplicationApplyOrdering` to 1.
- Set `ReplicationParallelism` to a number from 1 to 64. This number indicates the number of transmitter threads on the source database and the number of receiver threads on the target database. The default is 1, which indicates a single thread. Parallel replication requires at least two threads.

In addition, the application needs to assign transactions to tracks by one of these methods:

- Set the `ReplicationTrack` general connection attribute to a non-zero number. All transactions issued by the connection are assigned to this track. The value can be any number. TimesTen maps the `ReplicationTrack` number for this connection to one of the available parallel replication threads. Thus, the application can use any number to group transactions that should be applied in order. See "ReplicationTrack" in *Oracle TimesTen In-Memory Database Reference*.

- Use the `ALTER SESSION` SQL statement to set the replication track number for the current connection. See "ALTER SESSION" in *Oracle TimesTen In-Memory Database SQL Reference*.
- Use the `TT_REPLICATION_TRACK` ODBC connection option for the `SQLSetConnectOption` ODBC function. See "Features for use with replication" in *Oracle TimesTen In-Memory Database C Developer's Guide*
- Use the `setReplicationTrack()` method of the `TimesTenConnection` JDBC class. See "Features for use with replication" in *Oracle TimesTen In-Memory Database Java Developer's Guide*

Use the `ttConfiguration` built-in procedure to return the replication track number for the current connection. Use the `ttLogHolds` built-in procedure to verify that multiple tracks are being used.

Restrictions on user-defined parallel replication

- Do not configure user-defined parallel replication for tables that have an aging policy defined.
- Databases configured for user-defined parallel replication cannot contain cache groups.
- A database cannot be defined as a propagator when user-defined parallel replication is configured.
- User-defined parallel replication is not supported for synchronous replication, including databases with the `RETURN RECEIPT` and `RETURN TWOSAFE` attributes.
- Cross-release replication and migration from a database that does not have user-defined parallel replication enabled to a database that does have user-defined parallel replication enabled is not supported from release 11.2.1.6.0 until 11.2.1.8.0. It is supported from releases earlier than 11.2.1.6.0 and from 11.2.1.8.0 and later. Users of releases from 11.2.1.6.0 to 11.2.1.8.0 can perform an upgrade by first applying an in-place patch release upgrade to 11.2.1.8.0. For details, see "Performing an upgrade when using parallel replication" in the *Oracle TimesTen In-Memory Database Installation Guide*.

Managing the transaction log on a replicated database

This section includes these topics:

About log buffer flushing

A dedicated subdaemon thread writes the contents of the log buffer to disk periodically. These writes may be synchronous or buffered. The subdaemon thread ensures that the system I/O buffer never fills up with more transaction log data than the value of the `LogFileSize` first connection attribute without being synchronized to the log buffer.

If the database is configured with `LogFlushMethod=2`, then all writes to the disk are synchronous writes and the data is durably written to disk before the write call returns. If the database is configured with `LogFlushMethod=1`, then the writes are buffered unless there is a specific request from an application for synchronous writes.

In addition to the periodic writes, an application can also trigger the subdaemon thread to write the buffer contents to disk. The following are cases where the application triggers a synchronous write to the disk:

- When a transaction that requested a durable commit is committed. A transaction can request a durable commit by calling the `ttDurableCommit` built-in procedure or by having the `DurableCommits` connection attribute set to 1.
- 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).
- When the replication agent periodically executes a durable commit, whether the master database is configured with `TRANSMIT DURABLE` or not.

Transactions are also written to disk durably when durable commits are configured as part of the return service failure policies and a failure has occurred.

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 transaction log growth on a master database

In databases that do not use replication, Transaction Log API (XLA), cache groups or incremental backup, unneeded records in the log buffer and unneeded transaction log files are purged each time a checkpoint is initiated, either by the automatic background checkpointing thread or by an application's call to the `ttCkpt` or `ttCkptBlocking` built-in procedures. In a replicated database, transactions remain in the log buffer and transaction log files until the master replication agent confirms they have been fully processed by the subscriber. Only then can the master consider purging them from the log buffer and transaction log files.

A master database transaction log can grow much larger than it would on an unreplicated database if there are changes to its subscriber state. When the subscriber is in the `start` state, the master can purge logged data after it receives confirmation that the information has been received by the subscriber. However, if a subscriber becomes unavailable or is in the `pause` state, the log on the master database cannot be flushed and the space used for logging can be exhausted. When the log space is exhausted, subsequent updates on the master database are aborted. Use the `ttLogHolds` built-in procedure to get information about replication log holds.

For more information about transaction log growth, see "Monitoring accumulation of transaction log files" in *Oracle TimesTen In-Memory Database Operations Guide*.

Setting connection attributes for logging

`LogBufMB` specifies the maximum size of the in-memory log buffer in megabytes. This buffer is flushed to a transaction log file on the disk when it becomes full. The minimum size for `LogBufMB` is 8 times the value of `LogBufParallelism`.

You need to establish enough disk space for the transaction log files. There are two settings that control the amount of disk space used by the log:

- The `LogFileSize` setting in the DSN specifies the maximum size of a transaction log file. If logging requirements exceed this value, additional transaction log files with the same maximum size are created. If you set the `LogFileSize` to a smaller value than `LogBufMB`, TimesTen automatically increases the `LogFileSize` to match `LogBufMB`. For best performance, set `LogBufMB` and `LogFileSize` to their maximum values.

- The *log failure threshold* setting specifies the maximum number of transaction log files allowed to accumulate before the master assumes a subscriber has failed. The threshold value is the number of transaction log files between the most recently written to transaction log file and the earliest transaction 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 transaction 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 9-24 for more information.

Because transactions are logged to disk, you can use bookmarks to detect the log record identifiers 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 `ttBookmark` built-in procedure, as described in "Show replicated log records" on page 12-9.

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 transaction 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 database and start over, as described in Chapter 11, "Managing Database Failover and Recovery".

See *Oracle TimesTen In-Memory Database Reference* for more information about TimesTen connection attributes, built-in procedures and utilities.

Applying a replication scheme to a database

Define the replication scheme as described in Chapter 9, "Defining Replication Schemes". Save the `CREATE REPLICATION` statement in a SQL file.

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

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

Example 10–1 Creating a replication scheme by executing a SQL file

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 the replicated databases. You must invoke a separate `ttIsql` command on each host to apply the replication scheme.

Example 10–2 Executing a SQL file on each host

If your scheme includes the databases `masterDSN` on host `S1`, `subscriber1DSN` on host `S2`, and `subscriber2DSN` on host `S3`, do the following:

On host `S1`, enter:

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

On host `S2`, enter:

```
> ttIsql -f repscheme.sql -connstr "dsn=subscriber1DSN"
```

On host S3, enter:

```
> ttIsql -f repscheme.sql -connstr "dsn=subscriber2DSN"
```

You can also execute the SQL file containing your replication scheme from the ttIsql command line after connecting to a database. For example:

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

Duplicating a master database to a subscriber

The simplest method for populating a subscriber database is to duplicate the contents of the master database. Duplicating a database in this manner is also essential when recovering a failed database, as described in Chapter 11, "Managing Database Failover and Recovery". You can use the ttRepAdmin -duplicate utility or the ttRepDuplicateEx C function to duplicate a database.

To duplicate a database, these conditions must be fulfilled:

- The instance administrator performs the duplicate operation.
- The instance administrator user name must be the same on both instances involved in the duplication.
- You must provide the user name and password for a user with the ADMIN privilege on the source database.
- The target DSN cannot include client/server attributes.

To duplicate the contents of a master database to a subscriber database, complete these tasks:

1. Create or alter a replication scheme to include the new subscriber database and its host. See "Defining a replication scheme" on page 9-5 or "Creating and adding a subscriber database" on page 13-5.
2. Apply the replication scheme to the master database. See "Applying a replication scheme to a database" on page 10-11.
3. Start the replication agent for the master database. See "Starting and stopping the replication agents" on page 10-13.
4. On the source database (the master), create a user and grant the ADMIN privilege to the user:

   ```
   CREATE USER ttuser IDENTIFIED BY ttuser;
   User created.

   GRANT admin TO ttuser;
   ```

5. Assume the user name of the instance administrator is timesten. Logged in as timesten on the target host (the subscriber), duplicate database masterDSN on host1 to subscriber1DSN:

   ```
   ttRepAdmin -duplicate -from masterDSN -host host1 subscriber1DSN

   Enter internal UID at the remote datastore with ADMIN privileges: ttuser
   Enter password of the internal Uid at the remote datastore:
   ```

   Enter ttuser when prompted for the password of the internal user at the remote database.

Note: The host entry 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 (`host1` in this example) by using the `-localhost` option.

You can specify the local and remote network interfaces for the source and target hosts by using the `-localIP` and `-remoteIP` options of `ttRepAdmin -duplicate`. If you do not specify one or both network interfaces, TimesTen chooses them.

For details, see "ttRepAdmin" in *Oracle TimesTen In-Memory Database Reference*.

6. Start the replication agent on the subscriber database.

Configuring a large number of subscribers

A replication scheme can include up to 128 subscribers. A replication scheme with propagator databases can have up to 128 propagators, and each propagator can have up to 128 subscribers. An active standby pair replication scheme 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 database spawns a thread for every subscriber database. Each thread reads and processes the log independently and needs adequate CPU resources to transmit to the subscriber database.

Replicating databases across releases

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

For example, replication between a database created in TimesTen release 6.0 and a database created in TimesTen release 11.2.1 is not supported. However, if one database was created in TimesTen release 6.0, and the peer database was created in TimesTen release 6.0 and then upgraded to TimesTen release 11.2.1, replication between them is supported.

See "Database Upgrades" in *Oracle TimesTen In-Memory Database Installation Guide*.

Starting and stopping the replication agents

After you have defined a replication scheme, you can start the replication agents for each database involved in the replication scheme. You must have the `ADMIN` privilege to start or stop a replication agent.

You can start and stop a replication agent by using the `ttAdmin` utility with the `-repStart` or `-repStop` option. You can also use the `ttRepStart` and `ttRepStop`

built-in procedures to start and stop a replication agent from the `ttIsql` command line.

Example 10–3 Starting and stopping the replication agent with ttAdmin

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
```

Example 10–4 Starting and stopping the replication agent from ttIsql

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 database to `always` or `norestart`.

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

Note: The TimesTen daemon manages the replication agents. It must be running to start or stop the replication agents.

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

Example 10–5 Using ttAdmin to set the restart policy

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 a database 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 database. This happens after a fatal error that, for example, requires all applications to disconnect. The first connection to a database

usually has to load the most recent checkpoint file and often needs to do recovery. For a very large database, this process may take several minutes. During this period, all activity on the database 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 database existing at the same time because the old one stays around until all applications have disconnected. For very large databases for which the first-connect time may be significant, you may want to wait for the old database 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. For more information on setting policies that would prevent the database from being reloaded, see "Specifying a RAM policy" in *Oracle TimesTen In-Memory Database Operations Guide* to determine when a database is loaded into memory.

Setting the replication state of subscribers

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

- From the command line, use `ttRepAdmin -state` to direct a master database to reset the replication state of one of its subscriber databases.
- From ttIsql, call the `ttRepSubscriberStateSet` built-in procedure to direct a master database to reset the replication state of one or all of its subscriber databases.

See "Monitoring Replication" on page 12-1 for information about querying the state of a database.

A master database can set a subscriber database to either the `start`, `pause`, or `stop` states. The database state appears as an integer value in the `STATE` column in the `TTREP.REPPEERS` table, as shown in Table 10–2.

Table 10–2 Database states

State	Description
`start` `STATE` value: 0	Replication updates are collected and transmitted to the subscriber database as soon as possible. If replication for the subscriber database is not operational, the updates are saved in the transaction 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 database. Placing a subscriber database 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` database receive a warning. See "General failover and recovery procedures" on page 11-1 for more information.

When a master database 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 database. 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 Chapter 11, "Managing Database Failover and Recovery".) If a master database is unable to establish a connection to a stated subscriber, the master periodically attempts to establish a connection until successful.

Example 10–6 Using ttRepAdmin to set the subscriber state

To use `ttRepAdmin` from the command line to direct the `masterds` master database to set the state of the `subscriberds` subscriber database 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 `-host` option of the `ttRepAdmin` utility to identify the host for the subscriber that you want to modify.

Example 10–7 Using ttRepSubscriberStateSet to set the subscriber state

On the master database, call the `ttRepSubscriberStateSet` built-in procedure to set the state of the subscriber database (`subscriberds ON system1`) in the `repscheme` replication scheme to `stop`:

```
Command> CALL ttRepSubscriberStateSet('repscheme', 'repl',
          'subscriberds', 'system1', 2);
```

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.

11

Managing Database Failover and Recovery

This chapter applies to all replication schemes, including active standby pairs. However, TimesTen integration with Oracle Clusterware is the best way to monitor active standby pairs. See Chapter 7, "Using Oracle Clusterware to Manage Active Standby Pairs".

This chapter includes these topics:

- Overview of database failover and recovery
- General failover and recovery procedures
- Recovering a failed database
- Recovering nondurable databases
- Writing a failure recovery script

Overview of database failover and recovery

A fundamental element in the design of a highly available system is the ability to recover quickly from a failure. Failures may be related to hardware problems such as system failures or network failures. Software failures include operating system failure, application failure, database failure and 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 database, redirect the user load to one of its subscribers. 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 database. If a subscriber database is inoperable or communication to a subscriber database 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: The procedures described in this chapter require the `ADMIN` privilege.

General failover and recovery procedures

The procedures for managing failover and recovery depend primarily on:

- The replication scheme

- Whether the failure occurred on a master or subscriber database
- Whether the threshold for the transaction log on the master is exhausted before the problem is resolved and the databases reconnected

Subscriber failures

In a default asynchronous replication scheme, if a subscriber database becomes inoperable or communication to a subscriber database fails, 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 9-18.

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 9-24 for details on how to establish the `FAILTHRESHOLD` value for the master database.

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 database" on page 11-5. Any application that connects to the failed subscriber receives a `tt_ErrReplicationInvalid` (8025) warning indicating that the database has been marked `failed` by a replication peer.

An application can use the ODBC `SQLGetInfo` function to check if the subscriber database 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 database 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 11–1 Checking whether a database has been set to the failed state

Check if the database 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 database. If a master database fails, the cluster manager must detect this event and redirect the user load to one of its surviving databases. This surviving subscriber then becomes the master, which continues to accept transactions and replicates them to the other surviving subscriber databases. 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 database" on page 13-6 for an example.

When the problem is resolved, if you are not using the bidirectional configuration or the active standby pair described in "Automatic catch-up of a failed master database" on page 11-3, you must recover the master database as described in "Recovering a failed database" on page 11-5.

After the database 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."

Automatic catch-up of a failed master database

The master catch-up feature automatically restores a failed master database from a subscriber database without the need to invoke the `ttRepAdmin -duplicate` operation described in "Recovering a failed database" on page 11-5.

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 manner to a single subscriber
- An active standby pair that is configured with `RETURN TWOSAFE`

For replication schemes that are not active standby pairs, the following must be true:

- The `ELEMENT` type is `DATASTORE`.
- `TRANSMIT NONDURABLE` or `RETURN TWOSAFE` must be enabled.
- All replicated transactions must be committed nondurably. They must be transmitted to the remote database before they are committed on the local database. For example, if the replication scheme is configured with `RETURN TWOSAFE BY REQUEST` and any transaction is committed without first enabling `RETURN TWOSAFE`, master catch-up may not occur after a failure of the master.

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 (or from the standby to the active in an active standby pair). No connections are allowed to the master database until it has completely caught up with the subscriber. Applications attempting to connect to a database during the catch-up phase receive an error that indicates a catch-up is in progress. The only exception is connecting to a database with the `ForceConnect` first connection attribute set in the DSN.

When the catch-up phase is complete, the application can connect to the database. An SNMP trap and message to the system log indicate the completion of the catch-up phase.

If one of the databases is invalidated or crashes during the catch-up process, the catch-up phase is resumed when the database comes back up.

Master catch-up can fail under these circumstances:

- The failed database is offline long enough for the failure threshold to be exceeded on the subscriber database (the standby database in an active standby pair).
- Dynamic load operations are taking place on the active database in an active standby pair when the failure occurs. `RETURN TWOSAFE` is not enabled for dynamic load operations even though it is enabled for the active database. The database failure causes the dynamic load transactions to be trapped and `RETURN TWOSAFE` to fail.

When master catch-up is required for an active standby pair

TimesTen error 8110 (`Connection not permitted. This store requires Master Catchup.`) indicates that the standby database is ahead of the active database and that master catch-up must occur before replication can resume.

When using master catch-up with an active standby pair, the standby database must be failed over to become the new active database. If the old active database can recover, it becomes the new standby database. If it cannot recover, the old active database must be destroyed and the new standby database must be created by duplicating the new active database. See "When replication is return twosafe" on page 4-4 for more information about recovering from a failure of the active database when `RETURN TWOSAFE` is configured (required for master catch-up).

In an active standby pair with `RETURN TWOSAFE` configured, it is possible to have a *trapped transaction*. A trapped transaction occurs when the new standby database has a transaction present that is not present on the new active database after failover. Error 16227 (`Standby store has replicated transactions not present on the active`) is one indication of trapped transactions. You can verify the number of trapped transactions by checking the number of records in replicated tables on each database during the manual recovery process. For example, enter a statement similar to the following:

```
SELECT COUNT(*) FROM reptable;
```

When there are trapped transactions, perform these tasks for recovery:

1. Use the `ttRepStateSet` built-in procedure to change the state on the standby database to `'ACTIVE'`.
2. Destroy the old active database.
3. Use `ttRepAdmin -duplicate` to create a new standby database from the new active database, which has all of the transactions. See "Duplicating a database" on page 4-2.

Failures in bidirectional distributed workload schemes

You can distribute the workload over multiple bidirectionally replicated databases, each of which serves as both master and subscriber. When recovering a master/subscriber database, the log on the failed database may present problems when you restart replication. See "Bidirectional distributed workload scheme" on page 9-30.

If a database in a distributed workload scheme fails and work is shifted to a surviving database, the information in the surviving database becomes more current than that in the failed database. If replication is restarted at the failed system before the log failure threshold has been reached on the surviving database, then both databases 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 database 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 Chapter 14, "Resolving Replication Conflicts" are sufficient to guarantee consistency for your application, then you can restart the failed system and allow the updates from the failed database to propagate to the surviving database. The conflict resolution rules prevent more recent updates from being overwritten.

- Re-create the failed database, as described in "Recovering a failed database" on page 11-5. If the database must be re-created, the updates in the log on the failed database that were not received by the surviving database cannot be identified or restored. In the case of several surviving databases, you must select which of the surviving databases is to be used to re-create the failed database. It is possible that at the time the failed database is re-created, the selected surviving database may not have received all updates from the other surviving databases. This results in diverging databases. The only way to prevent this situation is to re-create the other surviving databases from the selected surviving database.

Network failures

In the event of a temporary network failure, you do not need to perform any specific action to continue replication. The replication agents that were in communication attempt to reconnect every few seconds. If the agents reconnect before the master database runs out of log space, the replication protocol makes sure they do not miss or repeat any replication updates. If the network is unavailable for a longer period and the log failure threshold has been exceeded for the master log, you need to recover the subscriber as described in "Recovering a failed database" on page 11-5.

Failures involving sequences

After a network link failure, if replication is allowed to recover by replaying queued logs, you do not need to take any action.

However, if the failed host was down for a significant amount of time, you must use the `ttRepAdmin -duplicate` command to repopulate the database on the failed host with transactions from the surviving host, as sequences are not rolled back during failure recovery. In this case, the `ttRepAdmin -duplicate` command copies the sequence definitions from one database to the other.

Recovering a failed database

If the databases are configured in a bidirectional replication scheme, a failed master database is automatically brought up to date from the subscriber. See "Automatic catch-up of a failed master database" on page 11-3. Automatic catch-up also applies to recovery of master databases in active standby pairs.

If a restarted database cannot be recovered from its master's transaction log so that it is consistent with the other databases in the replicated system, you must re-create the database from one of its replication peers. Use command line utilities or the TimesTen Utility C functions. See "Recovering a failed database from the command line" on page 11-6 and "Recovering a failed database from a C program" on page 11-6.

Note: It is not necessary to re-create the DSN for the failed database.

In the event of a subscriber failure, if any tables are configured with a return service, commits on those tables in the master database are blocked until the return service timeout 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 9-18. 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` statement to re-establish `RETURN RECEIPT`.

Recovering a failed database from the command line

If the databases are fully replicated, you can use the `ttDestroy` utility to remove the failed database from memory and `ttRepAdmin -duplicate` to re-create it from a surviving database. If the database contains any cache groups, you must also use the `-keepCG` option of `ttRepAdmin`. See "Duplicating a database" on page 4-2.

Example 11–2 Recovering a failed database

To recover a failed database, `subscriberds`, from a master named `masterds` on host `system1`, enter:

```
> ttdestroy /tmp/subscriberds

> ttrepadmin -dsn subscriberds -duplicate -from masterds -host "system1" -uid
ttuser
```

You will be prompted for the password of `ttuser`.

> **Note:** `ttRepAdmin -duplicate` is supported only between identical and patch TimesTen releases. The major and minor release numbers must be the same.

After re-creating the database with `ttRepAdmin -duplicate`, the first connection to the database reloads it into memory. To improve performance when duplicating large databases, you can avoid the reload step by using the `ttRepAdmin -ramload` option to keep the database in memory after the duplicate operation.

Example 11–3 Keeping a database in memory when recovering it

To recover a failed database, `subscriberds`, from a master named `masterds` on host `system1`, and to keep the database in memory and restart replication after the duplicate operation, enter:

```
> ttdestroy /tmp/subscriberds

> ttrepadmin -dsn subscriberds -duplicate -ramload -from masterds -host "system1"
-uid ttuser -setmasterrepstart
```

You will be prompted for the password of `ttuser`.

> **Note:** After duplicating a database with the `ttRepAdmin -duplicate -ramLoad` options, the RAM Policy for the database is `manual` until explicitly reset by `ttAdmin -ramPolicy` or the `ttRamPolicy` function.

Recovering a failed database from a C program

You can use the C functions provided in the TimesTen utility library to recover a failed database programmatically.

If the databases are fully replicated, you can use `ttDestroyDataStore` function to remove the failed database and the `ttRepDuplicateEx` function to re-create it from a surviving database.

Example 11–4 Recovering and starting a failed database

To recover and start a failed database, 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.uid=ttuser;
arg.pwd=ttuser;
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` database to the `start` state after the duplicate operation is completed
- `TT_REPDUP_RAMLOAD` to set the RAM policy to `manual` and keep the database in memory

> **Note:** When the `TT_REPDUP_RAMLOAD` flag is used with `ttRepDuplicateEx`, the RAM policy for the duplicate database is `manual` until explicitly reset by the `ttRamPolicy` function or `ttAdmin -ramPolicy`.

See "TimesTen Utility API" in *Oracle TimesTen In-Memory Database C Developer's Guide* for the complete list of the functions provided in the TimesTen C language utility library.

Recovering nondurable databases

If your database is configured with the `TRANSMIT NONDURABLE` option in a bidirectional configuration, you do not need to take any action to recover a failed master database. See "Automatic catch-up of a failed master database" on page 11-3.

For other types of configurations, if the master database configured with the `TRANSMIT NONDURABLE` option fails, you must use `ttRepAdmin -duplicate` or `ttRepDuplicateEx` to re-create the master database from the most current subscriber database. If the application attempts to reconnect to the master database without first performing the duplicate operation, the replication agent recovers the database, but any attempt to connect results in an error that advises you to perform the duplicate operation. To avoid this error, the application must reconnect with the `ForceConnect` first connection attribute 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 11–5.

Example 11–5 Failure recovery pseudocode

```
Detect problem {
       if (Master == unavailable) {
          FailedDataDatabase = Master
          FailedDSN = Master_DSN
          SurvivorDatabase = Subscriber
          switch users to SurvivorDatabase
      }
else {
          FailedDatabase = Subscriber
          FailedDSN = Subscriber_DSN
          SurvivorDatabase = Master
      }
}
Fix problem....
If (Problem resolved) {
       Get state for FailedDatabase
       if (state == "failed") {
         ttDestroy FailedDatabase
         ttRepAdmin -dsn FailedDSN -duplicate
                 -from SurvivorDatabase -host SurvivorHost
                 -setMasterRepStart
                 -uid ttuser
                 -pwd ttuser
      }
      else {
         ttAdmin -repStart FailedDSN
      }
      while (backlog != 0) {
         wait
      }
}

Switch users back to Master.
```

This applies to either the master or subscriber databases. If the master fails, you may lose some transactions.

12

Monitoring Replication

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

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 built-in procedures described in *Oracle TimesTen In-Memory Database Reference* or create your own SQL `SELECT` statements to query the replication tables described in *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

> **Note:** You can only access the TimesTen `SYS` and `TTREP` tables for queries. Do not try to alter the contents of these tables.

This chapter includes the following topics:

Show state of replication agents

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

You can also obtain the state of specific replicated databases as described in "Show subscriber database information" on page 12-4 and "Show the configuration of replicated databases" on page 12-6.

Using ttStatus to obtain replication agent status

Use the `ttStatus` utility to confirm that the replication agent is started for the master database.

Example 12–1 Using ttStatus to obtain replication agent status

```
> ttStatus
TimesTen status report as of Thu Aug 11 17:05:23 2011
Daemon pid 18373 port 4134 instance ttuser
TimesTen server pid 18381 started on port 4136
------------------------------------------------------------------------
Data store /tmp/masterds
There are 16 connections to the data store
Shared Memory KEY 0x0201ab43 ID 5242889
PL/SQL Memory KEY 0x0301ab43 ID 5275658 Address 0x10000000
Type            PID     Context     Connection Name             ConnID
Process         20564   0x081338c0  masterds                         1
Replication     20676   0x08996738  LOGFORCE                         5
Replication     20676   0x089b69a0  REPHOLD                          2
Replication     20676   0x08a11a58  FAILOVER                         3
Replication     20676   0x08a7cd70  REPLISTENER                      4
Replication     20676   0x08ad7e28  TRANSMITTER                      6
Subdaemon       18379   0x080a11f0  Manager                       2032
Subdaemon       18379   0x080fe258  Rollback                      2033
Subdaemon       18379   0x081cb818  Checkpoint                    2036
Subdaemon       18379   0x081e6940  Log Marker                    2035
Subdaemon       18379   0x08261e70  Deadlock Detector             2038
Subdaemon       18379   0xae100470  AsyncMV                       2040
Subdaemon       18379   0xae11b508  HistGC                        2041
Subdaemon       18379   0xae300470  Aging                         2039
Subdaemon       18379   0xae500470  Flusher                       2034
Subdaemon       18379   0xae55b738  Monitor                       2037
Replication policy  : Manual
Replication agent is running.
Cache Agent policy  : Manual
PL/SQL enabled.
```

Using ttAdmin -query to confirm policy settings

Use the `ttAdmin` utility with the `-query` option to confirm the policy settings for a database, including the replication restart policy described in "Starting and stopping the replication agents" on page 10-13.

Example 12–2 Using ttAdmin to confirm policy settings

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

Using ttDataStoreStatus to obtain replication agent status

To obtain the status of the replication agents from a program, use the `ttDataStoreStatus` built-in procedure.

Example 12–3 Calling ttDataStoreStatus

Call `ttDataStoreStatus` to obtain the status of the replication agents for the `masterds` databases:

```
> 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.
```

The output from `ttDataStoreStatus` is similar to that shown for the `ttStatus` utility in "Using ttStatus to obtain replication agent status" on page 12-2.

Show master database information

You can display information for a master database:

- Using ttRepAdmin to display information about the master database
- Querying replication tables to obtain information about a master database

Using ttRepAdmin to display information about the master database

Use the `ttRepAdmin` utility with the `-self -list` options to display information about the master database:

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

Example 12–4 Using ttRepAdmin to display information about a master database

This example shows the output for the master database described in "Multiple subscriber schemes with return services and a log failure threshold" on page 9-28.

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

The following table describes the fields.

Field	Description
`host`	The name of the host for the database.
`port`	TCP/IP port used by a replication agent of another database to receive updates from this database. A value of 0 (zero) indicates replication has automatically assigned the port.
`name`	Name of the database
`Log file/Replication hold LSN`	Indicates the oldest location in the transaction log that is held for possible transmission to the subscriber. A value of `-1/-1` indicates replication is in the `stop` state with respect to all subscribers.

Querying replication tables to obtain information about a master database

Use the following SELECT statement to query the TTREP.TTSTORES and TTREP.REPSTORES replication tables to obtain information about a master database:

```
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;
```

This is the output of the SELECT statement for the master database described in "Multiple subscriber schemes with return services and a log failure threshold" on page 9-28. The fields are the host name, the replication port number, and the database name.

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

Show subscriber database information

Replication uses the TimesTen transaction log to retain information that must be transmitted to subscriber sites. When communication to subscriber databases 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 database and the amount of time since the last successful communication with each subscriber database.

You can display information for subscriber databases:

- Using ttRepAdmin to display subscriber status
- Using ttReplicationStatus to display subscriber status
- Querying replication tables to display information about subscribers

Using ttRepAdmin to display subscriber status

To display information about subscribers, use the ttRepAdmin utility with the -receiver -list options:

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

Example 12–5 Using ttRepAdmin to display information about subscribers

This example shows the output for the subscribers described in "Multiple subscriber schemes with return services and a log failure threshold" on page 9-28.

```
> 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       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 transaction log files being retained on behalf of this subscriber. The latency for `subscriber1ds` is 19.41 seconds, and it is 2 logs behind the master. This is a high latency, indicating a problem if it continues to be high and the number of logs continues to increase.

If you have more than one scheme specified in the `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
```

Using ttReplicationStatus to display subscriber status

You can obtain more detailed status for a specific replicated database by using the `ttReplicationStatus` built-in procedure.

Querying replication tables to display information about subscribers

To obtain information about a master's subscribers from a program, use the following `SELECT` statement to query the `TTREP.REPPEERS`, `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 = t1.tt_store_id
    AND t2.is_local_store = 0X01
    AND p.subscriber_id = t2.tt_store_id
    AND p.replication_name = 'repscheme'
    AND p.replication_owner = 'repl'
    AND (p.state = 0 OR p.state = 1);
```

The following is sample output from the 3 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 `SELECT` statement contains the following fields:

Field	Description
`Peer name`	Name of the subscriber database
`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 database (see "Show subscriber database information" on page 12-4 for information).
`Protocol`	Internal protocol used by replication to communicate between this master and its subscribers. You can ignore this value.
`Last message sent`	Time (in seconds) since the master sent the last message to the subscriber. This includes the "heartbeat" messages sent between the databases.

Field	Description
`Last message received`	Time (in seconds) since this subscriber received the last message from the master.
`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.)
`Transactions per second`	The average number of transactions per second that are committed on the master and processed by the subscriber. (See note below.)
`Records per second`	The average number of transmitted records per second. (See note below.)
`Logs`	Number of transaction log files the master database 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 the configuration of replicated databases

You can display the configuration of your replicated databases:

- Using the ttIsql repschemes command to display configuration information
- Using ttRepAdmin to display configuration information
- Querying replication tables to display configuration information

Using the ttIsql repschemes command to display configuration information

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

```
Command> repschemes;
```

Example 12–6 shows the configuration output from the replication scheme shown in "Propagation scheme" on page 9-30.

Example 12–6 Output from ttIsql repschemes command

```
Replication Scheme PROPAGATOR:

  Element: A
    Type: Table TAB
    Master Store: CENTRALDS on FINANCE Transmit Durable
    Subscriber Store: PROPDS on NETHANDLER

  Element: B
    Type: Table 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
```

Using ttRepAdmin to display configuration information

To display the configuration of your replicated databases, use the `ttRepAdmin` utility with the `-showconfig` option:

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

Example 12–7 shows the configuration output from the propagated databases configured by the replication scheme shown in "Propagation scheme" on page 9-30. The `propds` propagator shows a latency of 19.41 seconds and is 2 logs behind the master.

Example 12–7 ttRepAdmin output

```
> 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 : tab          Timestamp updates : -

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

```
Table details
-------------
Table : tab          Timestamp updates : -

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

See "Querying replication tables to display information about subscribers" on page 12-5 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 databases.

Querying replication tables to display configuration information

Use the following `SELECT` statements to query the `TTREP.TTSTORES`, `TTREP.REPSTORES`, `TTREP.REPPEERS`, `SYS.MONITOR`, `TTREP.REPELEMENTS`, and `TTREP.REPSUBSCRIPTIONS` tables for configuration information:

```
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;
```

Example 12–8 Output from queries

The output from the queries refer to the databases configured by the replication scheme shown in "Propagation scheme" on page 9-30.

The output from the first query might be:

```
< finance, 0, centralds, 120, 0 >
```

It shows the host name, port number and the database name. The fourth value (120) is the `TIMEOUT` value that defines the amount of time a database waits for a response from another database before resending a message. The last value (0) is the log failure threshold value described in "Setting the log failure threshold" on page 9-24.

The output from the second query might be:

```
< propds, nethandler, 0, 0, 7, 1004378953, 0, -1.00000000000000, -1, -1, 1 >
```

See "Querying replication tables to display information about subscribers" on page 12-5 for a description of the fields.

The output from the last query might be:

```
< repl, tab, centralds, propds >
< repl, tab, propds, backup1ds >
< repl, tab, propds, backup2ds >
```

The rows show the replicated table and the names of its master (sender) and subscriber (receiver) databases.

Show replicated log records

In a replicated database, transactions remain in the log buffer and transaction log files until the master replication agent confirms they have been fully processed by the subscriber. Only then can the master consider purging them from the log buffer and transaction log files. When the log space is exhausted, subsequent updates on the master database are aborted. Use the `ttLogHolds` built-in procedure to get information about replication log holds. For more information about transaction log growth, see "Monitoring accumulation of transaction log files" in *Oracle TimesTen In-Memory Database Operations Guide*.

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 database.

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 amount of data 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 transaction 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.REPPEERS` 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. Because replication acknowledgements are asynchronous for better performance, the send LSN can also be some distance behind. Nonetheless, 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:

- Using ttRepAdmin to display bookmark location
- Using ttBookMark to display bookmark location

Using ttRepAdmin to display bookmark location

Use the `ttRepAdmin` utility with the `-bookmark` option to display the location of bookmarks:

```
> 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 database is not a master database).
`Last written LSN`	The location of the most recently generated transaction log record for the database.
`Last LSN forced to disk`	The location of the most recent transaction log record written to the disk.

Using ttBookMark to display bookmark location

Use the `ttBookmark` built-in procedure to display the location of bookmarks.

Example 12–9 Using ttBookmark to display bookmark location

```
> 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."

Using ttRepAdmin to 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 database.

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 database has a `TRANSMITTER` thread and a subscriber database has a `RECEIVER` thread. A database 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 database, the `REPLISTENER` thread starts a separate `TRANSMITTER` thread for each subscriber database. On a subscriber database, 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 12–9 shows `ttRepAdmin -showstatus` output for a unidirectional replication scheme in which the `rep1` database is the master and `rep2` database is the subscriber. The first `ttRepAdmin -showstatus` output shows the status of the `rep1` database and its `TRANSMITTER` thread. The second output shows the status of the `rep2` database 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 12–10 Unidirectional replication scheme

Consider the unidirectional replication scheme from the `rep1` database to the `rep2` database:

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

The replication status for the `rep1` database 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
```

Note that the `Replication hold LSN`, the `Last write LSN` and the `Last LSN` forced to disk are very close, which indicates that replication is operating satisfactorily. If the `Replication hold LSN` falls behind the `Last write LSN` and the `Last LSN`, then replication is not keeping up with updates to the master.

The replication status for the `rep2` database 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 database.

MAIN Thread	Description
`DSN`	Name of the database 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 10-13
`Host`	Name of the machine that hosts this database.

MAIN Thread	Description
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 database. See "Show replicated log records" on page 12-9 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 12-9 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 12-9 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 database. 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 database that is a replication peer to this database.
Host	Host of the peer database.
Port	TCP/IP port used by the replication agent for the peer database. 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 State	Current replication state of the replication peer with respect to the queried database (see "Show subscriber database information" on page 12-4 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 database that is receiving replicated data from this database.

TRANSMITTER Thread	Description
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 12-9 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)
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 database that is sending replicated data from this database
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.

RECEIVER Thread	Description
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)

Checking the status of return service transactions

You can determine whether the return service for a 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 12–11 Using ttRepSyncSubscriberStatus to obtain return receipt status

To use ttRepSyncSubscriberStatus to obtain the return receipt status of the subscriberds database with respect to its master database, 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 9-21 for more information.

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 database. 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 database, 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 9-15.

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 12–12 Reporting the status of each subscriber

You can call the `ttRepXactTokenGet` and `ttRepXactStatus` built-in procedures 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;
}
```

Improving replication performance

To increase replication performance, consider these tips:

- Configure parallel replication. See "Configuring parallel replication" on page 10-7.
- Use asynchronous replication, which is the default. For more information, see "Making decisions about performance and recovery tradeoffs" on page 9-3. However, if you are using active standby pairs, return twosafe (synchronous replication) has better performance than return receipt (semi-synchronous replication).
- Set the `LogFileSize` and `LogBufMB` first connection attributes to their maximum values. For more information, see "Setting connection attributes for logging" on page 10-10.
- If the workload is heavy enough that replication sometimes falls behind, replicated changes must be captured from the transaction logs on disk rather than from the in-memory log buffer. Using the fastest possible storage for the TimesTen transaction logs reduces I/O contention between transaction log flushing and replication capture and helps replication to catch up more quickly during periods of reduced workload. Consider using a high performance, cached disk array using a RAID-0 stripe across multiple fast disks or solid state storage.
- Experiment with the number of connections to the database where the updates are applied. If you need more than 64 concurrent connections, set the `Connections`

first connection attribute to a higher value. See "Connections" in *Oracle TimesTen In-Memory Database Reference*.

See also "Poor replication or XLA performance" in *Oracle TimesTen In-Memory Database Troubleshooting Guide*.

13

Altering Replication

This chapter describes how to alter an existing replication system. Table 13–1 lists the tasks often performed on an existing replicated system.

Table 13–1 Tasks performed on an existing replicated system

Task	What to do
Alter or drop a replication scheme	See"Altering a replication scheme" on page 13-1 and "Dropping a replication scheme" on page 13-8.
Alter a table used in a replication scheme	See "Altering a replicated table" on page 13-8.
Truncate a table used in a replication scheme	See "Truncating a replicated table" on page 13-8.
Change the replication state of a subscriber database	See "Setting the replication state of subscribers" on page 10-15.
Resolve update conflicts	See Chapter 14, "Resolving Replication Conflicts".
Recover from failures	See Chapter 11, "Managing Database Failover and Recovery".
Upgrade database	Use the `ttMigrate` and `ttRepAdmin` utilities, as described in "Database Upgrades" in *Oracle TimesTen In-Memory Database Installation Guide*.

Altering a replication scheme

You can perform the following tasks without stopping the replication agent:

- Create, alter or drop a user. These statements are replicated.
- Grant or revoke privileges from a user. These statements are replicated.
- Add a subscriber to the replication scheme. See "Creating and adding a subscriber database" on page 13-5.
- Add a PL/SQL object to the master database and implement its replication on subscribers. See "Adding a PL/SQL object to an existing replication scheme" on page 13-3.

Use `ALTER REPLICATION` to alter the replication scheme on the master and subscriber databases. Any alterations on the master database must also be made on its subscribers.

Note: You must have the ADMIN privilege to use the ALTER REPLICATION statement.

Most ALTER REPLICATION operations are supported only when the replication agent is stopped (ttAdmin -repStop). 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 databases. While the replication agents are stopped, changes to the master database are stored in the log.
2. Issue the same ALTER REPLICATION statement on both master and subscriber databases.
3. Use the ttRepStart procedure or ttAdmin -repStart to restart the replication agent for the master and subscriber databases. The changes stored in the master database log are sent to the subscriber database.

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

This section includes the following topics:

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 13–1.
- 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 13–2.

Example 13–1 Adding a sequence and a table to a replication scheme

This example uses the replication scheme r1, which was defined in Example 9–29. It alters replication scheme r1 to add sequence seq and table westleads, which will be updated on database westds and replicated to database eastds.

```
ALTER REPLICATION r1
  ADD ELEMENT elem_seq SEQUENCE seq
    MASTER westds ON "westcoast"
    SUBSCRIBER eastds ON "eastcoast"
  ADD ELEMENT elem_westleads TABLE westleads
    MASTER westds ON "westcoast"
    SUBSCRIBER eastds ON "eastcoast";
```

Example 13–2 Adding a sequence and a table to a DATASTORE element

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 PL/SQL object to an existing replication scheme

To add a new PL/SQL procedure, package, package body or function to an existing replication scheme, complete these tasks:

1. Create the PL/SQL object on a master database. The CREATE statement is not replicated to subscribers.
2. Create the PL/SQL object on the subscribers
3. Grant privileges to the new PL/SQL object on the master database. The GRANT statement is replicated to the subscribers.

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 views, and nonmaterialized views are included in the replication scheme if you do not use the INCLUDE or EXCLUDE clauses. See "Including tables or sequences when you add a DATASTORE element" on page 13-3 and "Excluding a table or sequence when you add a DATASTORE element" on page 13-4.

Example 13–3 Adding a DATASTORE element to a replication scheme

Add a DATASTORE element to an existing replication scheme.

```
ALTER REPLICATION my.rep1
  ADD ELEMENT ds1 DATASTORE
       MASTER rep2
       SUBSCRIBER rep1, rep3;
```

Including tables or sequences when you add a DATASTORE element

You can restrict replication to specific tables or sequences when you add a database to an existing replication scheme. Use the ALTER REPLICATION statement with the ADD ELEMENT clause and the INCLUDE TABLE clause or INCLUDE SEQUENCE clause. You

can have one INCLUDE clause for each table or sequence in the same ALTER REPLICATION statement.

Example 13–4 Including a table and sequence in a DATASTORE element

Add the ds1 DATASTORE element to my.rep1 replication scheme. Include the table my.tab2 and the sequence my.seq 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;
```

Excluding a table or sequence when you add a DATASTORE element

You can exclude tables or sequences 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 or EXCLUDE SEQUENCE clause. You can have one EXCLUDE clause for each table or sequence in the same ALTER REPLICATION statement.

Example 13–5 Excluding a table or sequence from a DATASTORE element

Add the ds2 DATASTORE element to a replication scheme, but exclude the table my.tab1 and the sequence my.seq.

```
ALTER REPLICATION my.rep1
ADD ELEMENT ds2 DATASTORE
MASTER rep2
SUBSCRIBER rep1
EXCLUDE TABLE my.tab1
EXCLUDE SEQUENCE my.seq;
```

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 13–6 Excluding a table from a DATASTORE element and then dropping the table

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 13–7 Dropping an element from a replication scheme and then dropping the sequence

Drop the `SEQUENCE` element `elem_seq` from the replication scheme `r1`. Then drop the sequence `seq`.

```
ALTER REPLICATION r1
  DROP ELEMENT elem_seq;
DROP SEQUENCE seq;
```

Creating and adding a subscriber database

You can add a new subscriber database while the replication agents are running. To add a database to a replication scheme, do the following:

1. Make sure the new subscriber database does not exist.
2. Apply the appropriate statements to all participating databases:

   ```
   ALTER REPLICATION ...
     ALTER ELEMENT ...
       ADD SUBSCRIBER ...
   ```

3. On the source database (the master), create a user and grant the `ADMIN` privilege to the user:

   ```
   CREATE USER ttuser IDENTIFIED BY ttuser;
   User created.

   GRANT admin TO ttuser;
   ```

4. Logged in as the instance administrator, run the `ttRepAdmin -duplicate` command to copy the contents of the master database 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.
5. Start the replication agent on the newly created database (`ttAdmin -repStart`).

Example 13–8 Adding a subscriber to a replicated table

This example alters the `r1` replication scheme to add a subscriber (`backup3`) to the `westleads` table (step 2 above):

```
ALTER REPLICATION r1
  ALTER ELEMENT elem_westleads
    ADD SUBSCRIBER backup3 ON "backupserver";
```

Dropping a subscriber database

Stop the replication agent before you drop a subscriber database.

This example alters the `r1` replication scheme to drop the `backup3` subscriber for the `westleads` table:

Example 13–9 Dropping a subscriber for a replicated table

```
ALTER REPLICATION 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 13–10 Changing a table name

```
ALTER REPLICATION 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 database

Stop the replication agent before you replace a master database.

In this example, `newwestds` is made the new master for all elements currently configured for the master, `westds`:

Example 13–11 Replacing a master database

```
ALTER REPLICATION 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 14–2 is eliminated for the `elem_accounts_1` table:

Example 13–12 Eliminating conflict detection for a table

```
ALTER REPLICATION r1
  ALTER ELEMENT elem_accounts_1
```

```
    SET NO CHECK;
```

See Chapter 14, "Resolving Replication Conflicts" 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 9–29:

Example 13–13 Eliminating return receipt service for a subscriber

```
ALTER REPLICATION 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 the replication agent of a subscriber database accepts connection requests from the master replication agent. See "Port assignments" on page 9-24 for details on how to assign port to the replication agents.

In this example, the `r1` replication scheme is altered to change the port number of the `eastds` to 22251:

Example 13–14 Changing a port number for a database

```
ALTER REPLICATION 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 13–15 Changing the replication route

In this example, the `rep.r1` replication scheme is altered to change the priority 2 IP address for the master database from 192.168.1.100 to 192.168.1.101:

```
ALTER REPLICATION 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;
```

Changing the log failure threshold

Use the `FAILTHRESHOLD` attribute of the `STORE` parameter to reset the log failure threshold. Stop the replication agents before using `ALTER REPLICATION` or `ALTER ACTIVE STANDBY PAIR` to define a new threshold value, and then restart the replication agents.

See "Setting the log failure threshold" on page 3-11 and "Setting the log failure threshold" on page 9-24 for more information about the log failure threshold.

Altering a replicated table

You can use `ALTER TABLE` to add or drop columns on the master database. The `ALTER TABLE` operation is replicated to alter the subscriber databases.

If you use `ALTER TABLE` on a database configured for bidirectional replication, first stop updates to the table on all of the replicated databases and confirm all replicated updates to the table have been received by the databases before issuing the `ALTER TABLE` statement. Do not resume updates until the `ALTER TABLE` operation has been replicated to all databases. This is necessary to ensure that there are no write operations until after the table is altered on all databases.

Note: You can use the `ttRepSubscriberWait` procedure or monitoring tools described in Chapter 12, "Monitoring Replication" to confirm the updates have been received and committed on the databases.

Also, if you are executing a number of successive `ALTER TABLE` operations on a database, 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 database. 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 database 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 database. You cannot drop a replication scheme when master catchup is required unless it is the only replication scheme in the database.

Note: You must have the `ADMIN` privilege to use the `DROP REPLICATION` statement.

You must stop the replication agent before you drop a replication scheme.

Example 13–16 Dropping a replication scheme

To remove the `repscheme` replication scheme from a database, enter the following:

```
DROP REPLICATION 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 13–17 Removing a table and a replication from a database

To remove the `tab` table and `repscheme` replication scheme from a database, enter the following:

```
DROP REPLICATION repscheme;
DROP TABLE tab;
```

14

Resolving Replication Conflicts

This chapter includes these topics:

How replication conflicts occur

Tables in databases configured in a bidirectional replication scheme may be subject to replication conflicts. A replication conflict occurs when applications on bidirectionally replicated databases initiate an update, insert or delete operation on the same data item at the same time. If no special steps are taken, each database can end up in disagreement with the last update made by the other database.

These types of replication conflicts can occur:

- *Update conflicts*: This type of conflict occurs when concurrently running transactions at different databases 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 databases 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 database deletes a row while a concurrent transaction at another database 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 "Reporting conflicts" on page 14-7 for example reports generated by TimesTen upon detecting update, uniqueness, and delete conflicts.

Note: TimesTen does not detect conflicts involving `TRUNCATE TABLE` statements.

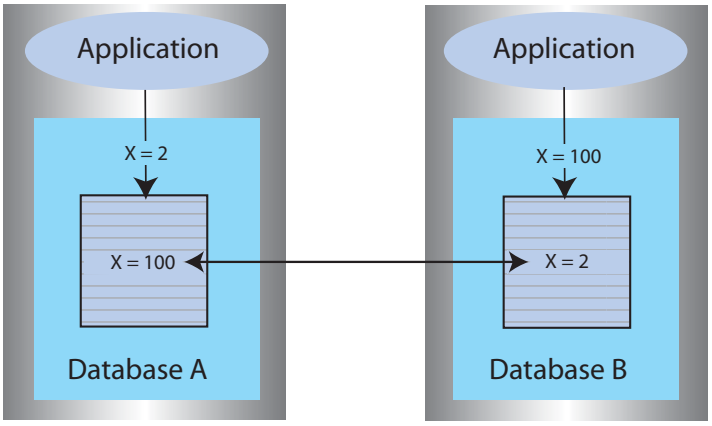
Update and insert conflicts

Figure 14–1 shows the results from an update conflict, which would occur for the value of X under the following circumstances:

Steps	On Database A	On Database B
Initial condition	X is 1.	X is 1.
The application on each database updates X simultaneously.	Set X=2.	Set X=100.
The replication agent on each database sends its update to the other database.	Replicate X to database B.	Replicate X to database A.
Each database 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, but the conflict involves the whole row.

Figure 14–1 Update conflict



If update or insert conflicts remain unchecked, the master and subscriber databases fall out of synchronization with each other. It may be difficult or even impossible to determine which database 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 database, update conflicts are prevented by the locking protocol: only one transaction at a time can update a specific row in the database. However, update conflicts can occur in replicated systems due to the ability of each database to operate independently.

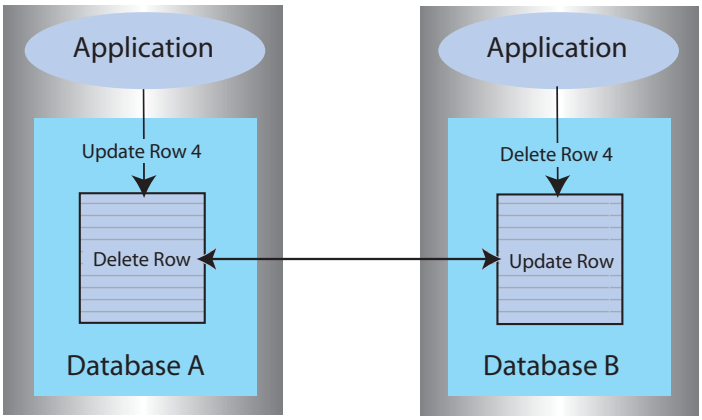
TimesTen replication uses timestamp-based conflict resolution to cope with simultaneous updates or inserts. Through the use of timestamp-based conflict resolution, you may be able to keep the replicated databases synchronized and transactionally consistent.

Delete/update conflicts

Figure 14–2 shows the results from a delete/update conflict, which would occur for Row 4 under the following circumstances:

Steps	On database A	On database 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 database sends the delete or update to the other	Replicate update to database B	Replicate delete to database A
Each database now has the delete or update from the other database	Replication says to delete Row 4	Replication says to update Row 4

Figure 14–2 Delete/update conflict



Although TimesTen can detect and report delete/update conflicts, it cannot resolve them. Under these circumstances, the master and subscriber databases fall out of synchronization with each other.

Although TimesTen cannot ensure synchronization between databases following such a conflict, it does ensure that the most recent transaction is applied to each database. If the timestamp for the delete is more recent than that for the update, the row is deleted on each database. If the timestamp for the update is more recent than that for the delete, the row is updated on the local database. However, because the row was deleted on the other database, the replicated update is discarded. See "Reporting delete/update conflicts" on page 14-11 for example reports.

Note: There is an exception to this behavior when timestamp comparison is enabled on a table using `UPDATE BY USER`. See "Enabling user timestamp column maintenance" on page 14-7 for details.

Using a timestamp to resolve conflicts

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 14-5.

> **Note:** TimesTen does not support conflict resolution between cached tables in a cache group and an Oracle database.

How replication computes the timestamp column depends on your system:

- 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.

- On Windows systems, the timestamp value is derived from the **GetSystemTimeAsFileTime** Win32 call. The Windows 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 database 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` clause is specified for a table, then the update, insert, or delete that fails a timestamp comparison is rejected. This may result in transactional inconsistencies if replication applies some, but not all, the actions of a transaction. If the `ON EXCEPTION ROLLBACK WORK` clause is specified for a table, an update that fails timestamp comparison causes the entire transaction to be skipped.

Timestamp comparisons for 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, but by protecting record timestamps from "going backwards," replication can help to ensure that the tables on replicated systems stay synchronized.

Configuring timestamp comparison

To configure timestamp comparison:

- Include a column in your replicated tables to hold the timestamp value. See "Including a timestamp column in replicated tables" on page 14-5.
- Include a `CHECK CONFLICTS` clause for each `TABLE` element in the `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. See "Configuring the CHECK CONFLICTS clause" on page 14-5.

Including 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 14–1 shows that the `rep.tab` table contains a column named `tstamp` of type `BINARY(8)` to hold the timestamp value.

Example 14–1 Including a timestamp column when creating a table

```
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 *Oracle TimesTen In-Memory Database SQL Reference*. Example 14–2 shows how CHECK CONFLICTS might be used when configuring your replication scheme.

Example 14–2 Automatic timestamp comparison

In this example, we establish automatic timestamp comparison for the bidirectional replication scheme defined in Example 9–29. The DSNs, west_dsn and east_dsn, define the westds and eastds databases that replicate the repl.accounts table containing the tstamp timestamp table. In the event of a comparison failure, discard the transaction that includes an update with the older timestamp.

```
CREATE REPLICATION r1
ELEMENT elem_accounts_1 TABLE 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 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 databases with conflict resolution, the replicated tables on each database 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 13-6 and apply to each replicated database.

Enabling system timestamp column maintenance

Enable system timestamp comparison by 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.

Enabling user timestamp column maintenance

Enable user timestamp column maintenance on a table by using:

```
CHECK CONFLICTS BY ROW TIMESTAMP
  COLUMN ColumnName
  UPDATE BY USER
```

When you configure `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.

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.

Reporting conflicts

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 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 databases 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_database
Transmitting name : master_database
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 14-9, "Reporting update conflicts" on page 14-10 and "Reporting delete/update conflicts" on page 14-11.

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 "The conflict report XML Document Type Definition" on page 14-13.

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> <<columnvalue> [,<columnvalue>...]>
End of failed transaction
```

Example 14–3 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 14–3 Output from uniqueness conflict

```
Conflict detected at 13:36:00 on 03-25-2002
Datastore : /tmp/masterds
Transmitting name : SUBSCRIBERDS
Table : 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 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 14–4 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` database conflicts with the newer update in `subscriberds`, so the replicated update is discarded.

Example 14–4 Output from an update conflict

```
Conflict detected at 15:03:18 on 03-25-2002
Datastore : /tmp/subscriberds
Transmitting name : MASTERDS
Table : 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 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 14–5 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 14–5 Output from a delete/update conflict: delete is more recent

```
Conflict detected at 15:27:05 on 03-25-2002
Datastore : /tmp/masterds
Transmitting name : SUBSCRIBERDS
Table : 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 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>
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 14–6 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 14–6 Output from a delete/update conflict: update is more recent

```
Conflict detected at 15:27:20 on 03-25-2002
Datastore : /tmp/subscriberds
Transmitting name : MASTERDS
Table : 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 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 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 can 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 "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.

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.

If you set `CONFLICT REPORTING RESUME AT` to `0`, reporting does not resume until the replication agent is restarted.

Example 14–7 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.

Example 14–7 Configuring conflict reporting thresholds

```
CREATE REPLICATION r1
ELEMENT elem_accounts_1 TABLE 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 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;
```

The conflict report XML Document Type Definition

The TimesTen XML format conflict report is are based on the XML 1.0 specification (http://www.w3.org/TR/REC-xml). 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, identified by 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, scope, failedtransaction) >
    <!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*, 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 that describes the report format and links 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 14-9 for a description of the information that is written to the conflict report for a uniqueness conflict.

Example 14–8 illustrates the format of a uniqueness conflict XML element, using the values from Example 14–3.

Example 14–8 Uniqueness conflict element

```
<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">
           <columname>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 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 14-10 for a description of the information that is written to the conflict report for an update conflict.

Example 14–9 illustrates the format of an update conflict XML element, using the values from Example 14–4.

Example 14–9 Update conflict element

```
<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</columname>
              <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 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 database receiving the update, or when a replicated deletion fails because the row has been updated more recently. See "Reporting delete/update conflicts" on page 14-11 for a description of the information that is written to the conflict report for a delete/update conflict.

Example 14–10 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 14–5.

Example 14–10 Delete/update conflict element: delete is more recent

```
<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 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 14–11 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 14–6.

Example 14–11 Delete/update conflict element: update is more recent

```
<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 TAB</sql>
 <keyinfo>
     <column pos="1">
       <columnname>COL1</columnname>
       <columntype>NUMBER(38)</columntype>
       <columnvalue>147</columnvalue>
     </column>
    </keyinfo>
   </delete>
 </failedtransaction>
</repconflict>
```

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W