Oracle® TimesTen In-Memory Database

Replication Guide

Release 26.1
F54450-01
March 2026

Oracle TimesTen In-Memory Database Replication Guide, Release 26.1

F54450-01

Contents

About This Content

This guide provides background information to help you understand how TimesTen works, as well as step-by-step instructions that show how to perform the most commonly-needed tasks..

Audience

To work with this guide, you should understand how database systems work and have some knowledge of Structured Query Language (SQL).

Conventions

The following text conventions are used in this document.

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

1

Overview of TimesTen Replication

TimesTen replication features are available with Oracle TimesTen In-Memory Database in classic mode (TimesTen Classic).

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.

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 with all of its contents between databases is called a replication element. TimesTen Classic supports databases, cache groups, tables and sequences as replication elements. TimesTen Classic also supports replicating XLA bookmarks.

The active standby pair configuration provides the highest availability. In an active standby pair replication scheme, the data is copied from the active database to the standby database before potentially being copied to read-only subscribers.

- An active standby pair is the only supported replication scheme for databases with cache groups.
- Certain DDL statements in an active standby pair are replicated against the other nodes in the replication scheme. See Making DDL Changes in an Active Standby Pair.

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.

Starting with Release 26.1, on Linux and UNIX, TimesTen has strengthened password security by adopting a stronger hashing algorithm. For replication between different major versions of TimesTen, the Times databases (on all the replication peers) must be one of the following releases to support the stronger algorithm and avoid authentication issues:

- Release 26.1.1.1.0 or later for TimesTen 26.1
- Release 22.1.1.36.0 or later for TimesTen 22.1
- Release 18.1.4.54.0 and later for TimesTen 18.1

The databases must have DSNs with identical value in the `DatabaseCharacterSet` database connection attribute.

See Connection Attributes for Replicated Databases.

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 a master database reads the records from the transaction log and forwards any detected changes to replicated elements to the replication agent on a subscriber database. The replication agent on a subscriber database applies the updates to its database. If the subscriber agent is not running when the updates are forwarded by the master, then the master retains the updates in the transaction 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 Replication Tables in *Oracle TimesTen In-Memory Database System Tables and Views Reference*.

You can enable secure TCP/IP network connections between replication agents (and utilities that communicate with the replication agents) by using Transport Layer Security (TLS), which requires mutual authentication to encrypt communication over connections. You can also generate certificates. See Transport Layer Security for TimesTen Replication in *Oracle TimesTen In-Memory Database Security Guide*.

Copying Updates Between Databases

By default, updates are copied between databases asynchronously.

While asynchronous replication provides the best performance, it does not provide the application with confirmation that the replicated updates were committed on 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 before being committed on the master.

Return receipt replication impacts performance less than return twosafe, but 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

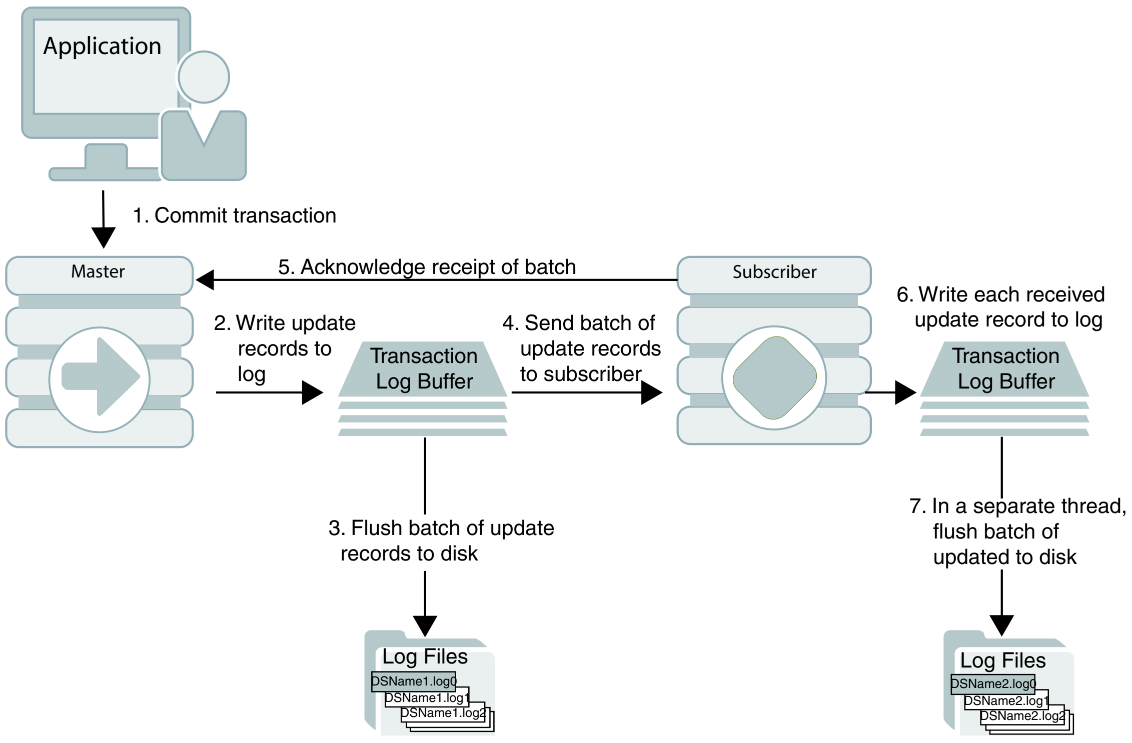
Default Replication

When using default TimesTen Classic 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 Classic 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


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

The default TimesTen Classic 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 the file system 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 the file system. The master replication agent is now free to purge from the transaction log the update records that have been received, applied, and flushed to the file system 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

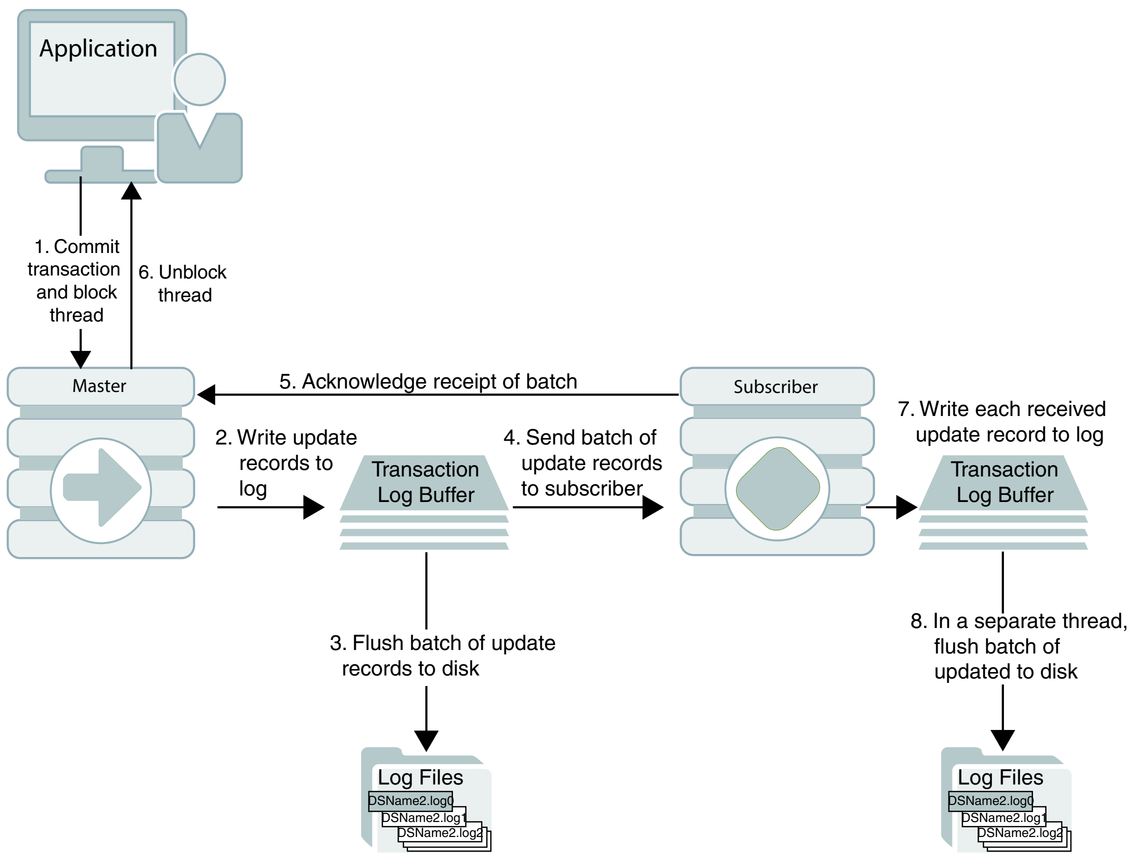

Application
1. Commit transaction and block thread
6. Unblock thread
Master
5. Acknowledge receipt of batch
Subscriber
2. Write update records to log
Transaction Log Buffer
4. Send batch of update records to subscriber
7. Write each received update record to log
Transaction Log Buffer
3. Flush batch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

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 processing 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.

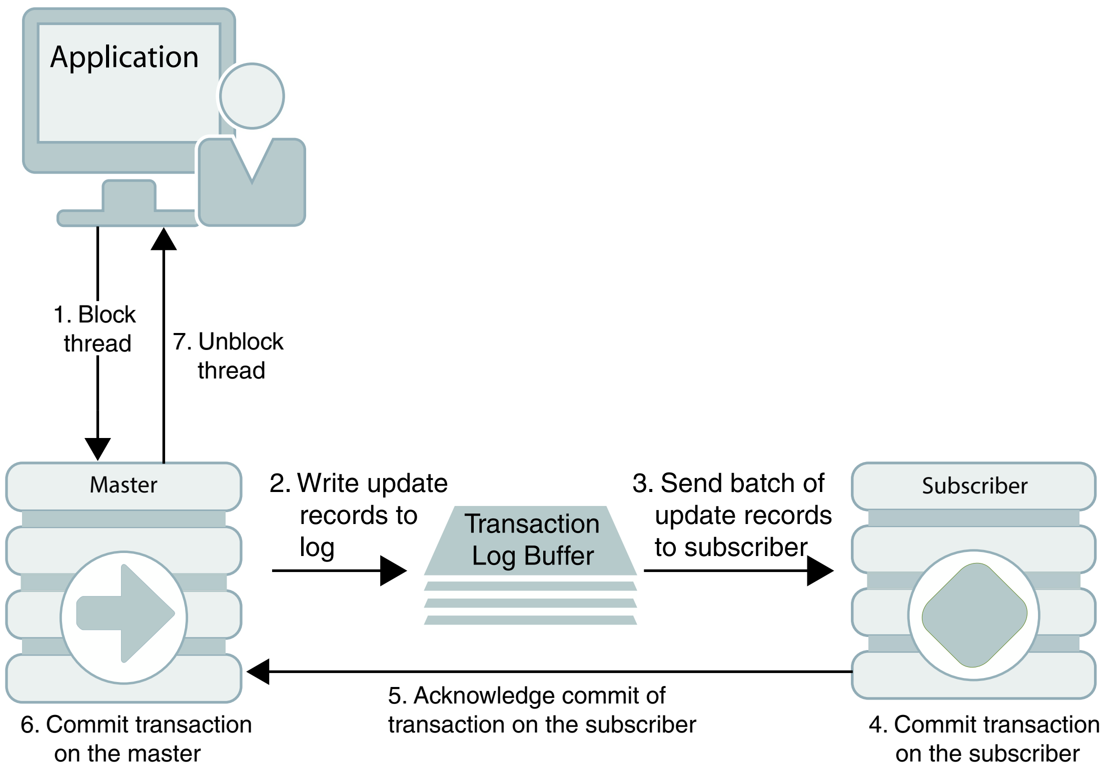
Return receipt transactions may time out for many reasons. The most likely causes for timeout are the network, a failed replication agent, or the master replication agent may be so far behind with respect to the transaction load that it cannot replicate the return receipt transaction before its timeout expires. See Setting the Return Service Timeout Period and 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


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

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 pre-commit log record before the commit record. This pre-commit 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 non-durable, so the master replication agent does not flush the log records to the file system 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.

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:

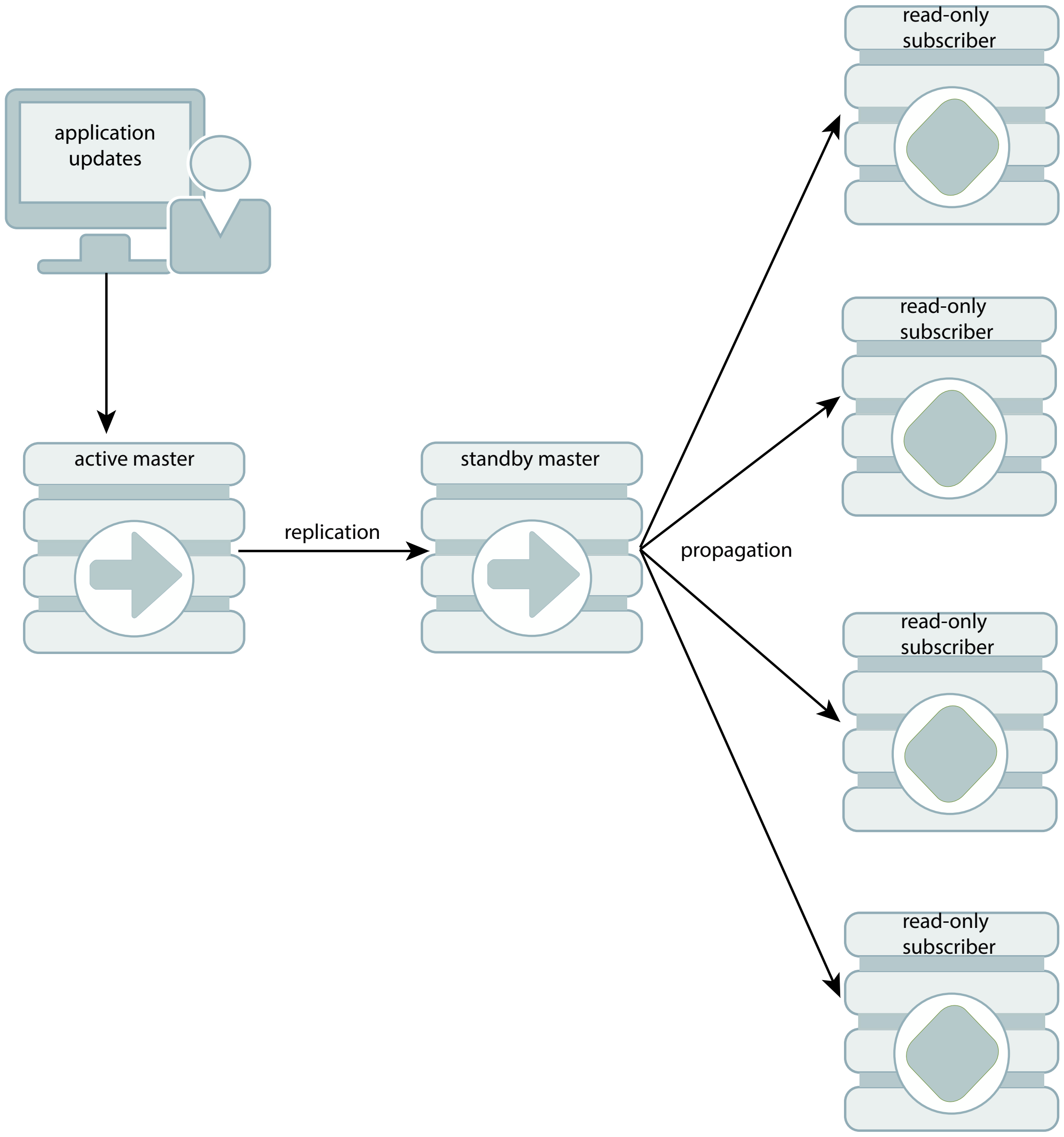
- Active Standby Pair With Read-Only Subscribers
- Classic Replication

Active Standby Pair With Read-Only Subscribers

You can create an active standby pair replication scheme with an active master, a standby master, and several read-only subscriber databases.

Figure 1-4 shows an active standby pair replication scheme with an active master, a standby master, 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 master databases. One is an active master, and the other is a standby master. The application updates the active master directly. Applications cannot update the standby master. It receives the updates from the active master and propagates the changes to as many as 127 read-only subscriber databases. This arrangement ensures that the standby master is always ahead of the subscriber databases and enables rapid failover to the standby master if the active master fails.

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

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

For details about setting up an active standby pair, see Defining an Active Standby Pair Replication Scheme, Setting Up an Active Standby Pair with No Cache Groups, Setting Up an Active Standby Pair with a Read-Only Cache Group, or Setting Up an Active Standby Pair with an AWT Cache Group.

Classic Replication

Classic replication schemes enable you to design relationships between masters and subscribers.

The following sections describe classic replication schemes:

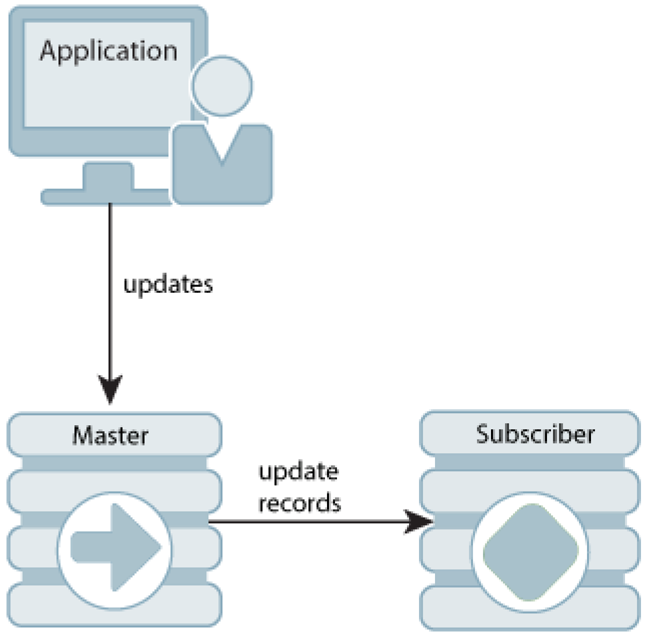
- Full Database Replication or Selective Replication
- Unidirectional or Bidirectional Replication
- Direct Replication or Propagation

Full Database Replication or Selective Replication

You can replicate a full master database or selectively replicate some elements in the master database to a subscriber database.

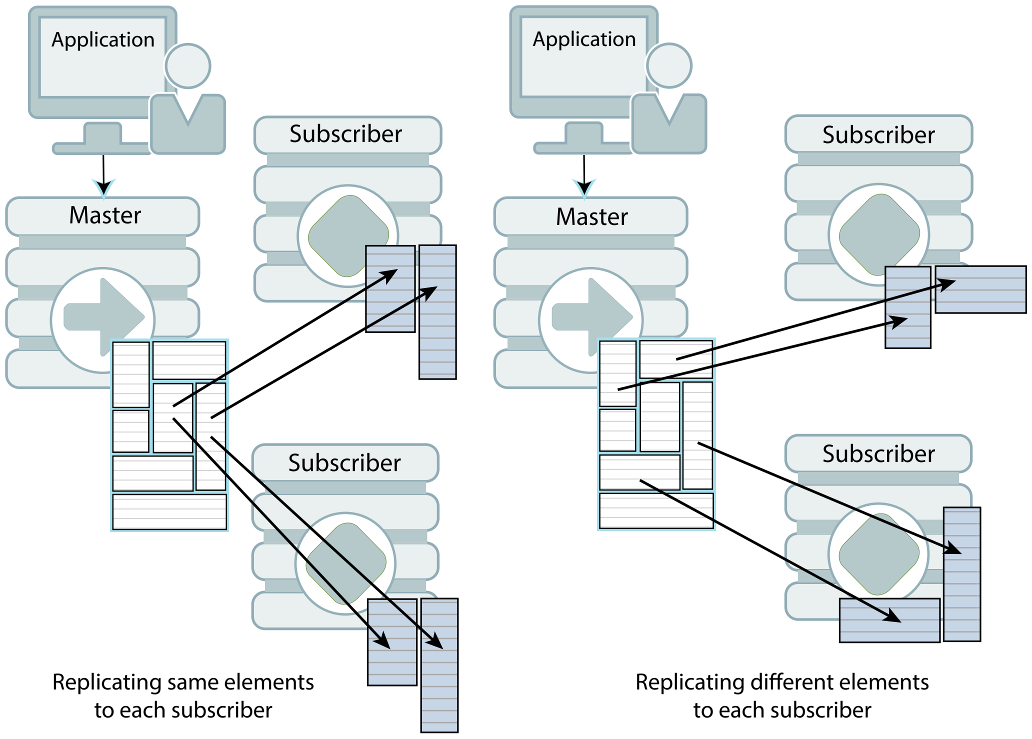
Figure 1-5 illustrates a full replication scheme in which the entire master database is replicated to the subscriber.

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


Application
Subscriber
Master
Subscriber
Replicating same elements to each subscriber
Application
Subscriber
Master
Subscriber
Replicating different elements to each subscriber

Unidirectional or Bidirectional Replication

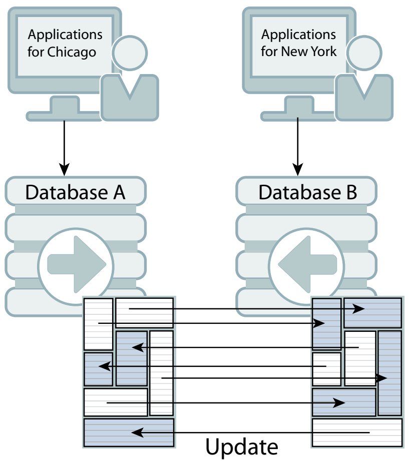
Unidirectional replication consists of a master database that sends updates to one or more subscriber databases. Bidirectional replication consists of two databases that operate bidirectionally, where each database is both a master and a subscriber to each other.

These are basic ways to use bidirectional replication:

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

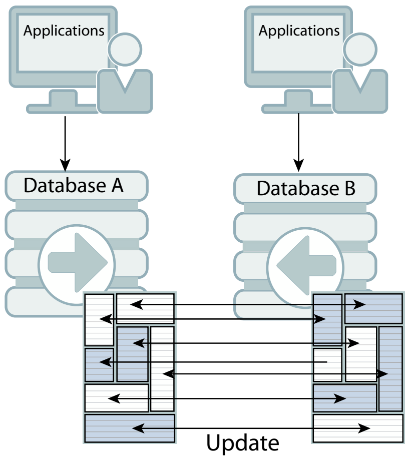
  Consider the example shown in Figure 1-7, where the accounts for Chicago are processed on database A while the accounts for New York are processed on database B.

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 Resolving Replication Conflicts.

Note

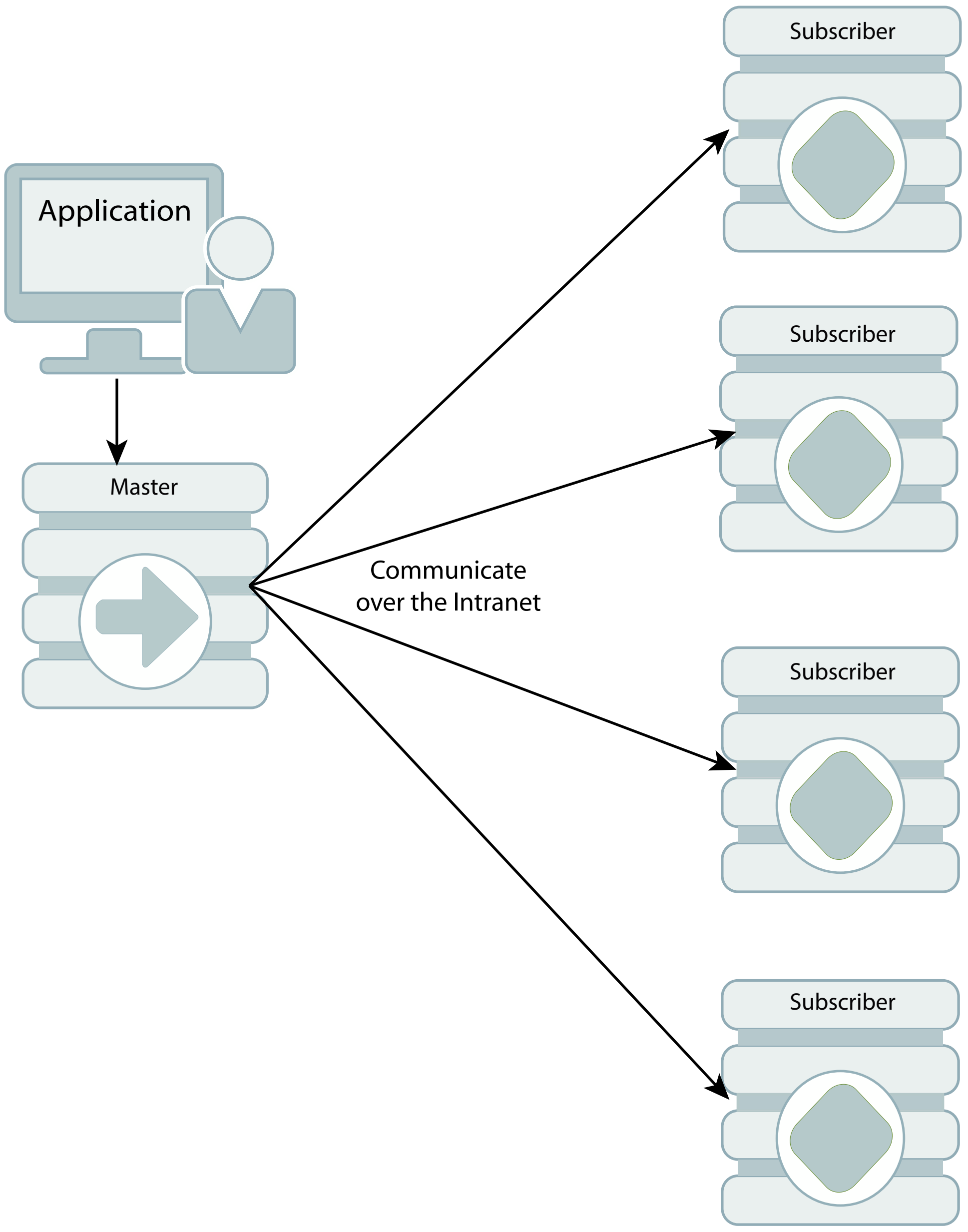
Do not use a distributed workload configuration with the return twosafe return service.

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.

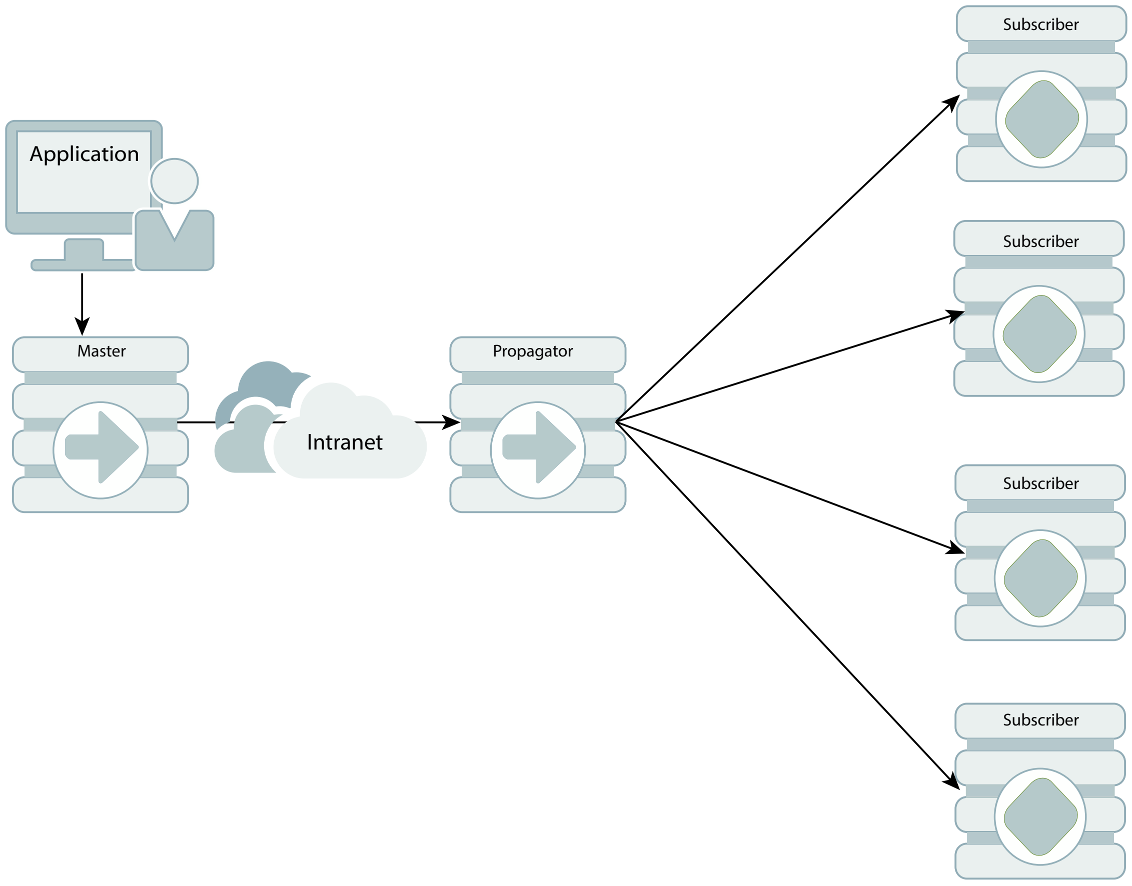
Propagators are useful for optimizing replication performance over lower-bandwidth network connections, such as those between servers in an intranet. For example, consider the direct replication configuration illustrated in Figure 1-9, where a master directly replicates to four subscribers over an intranet 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



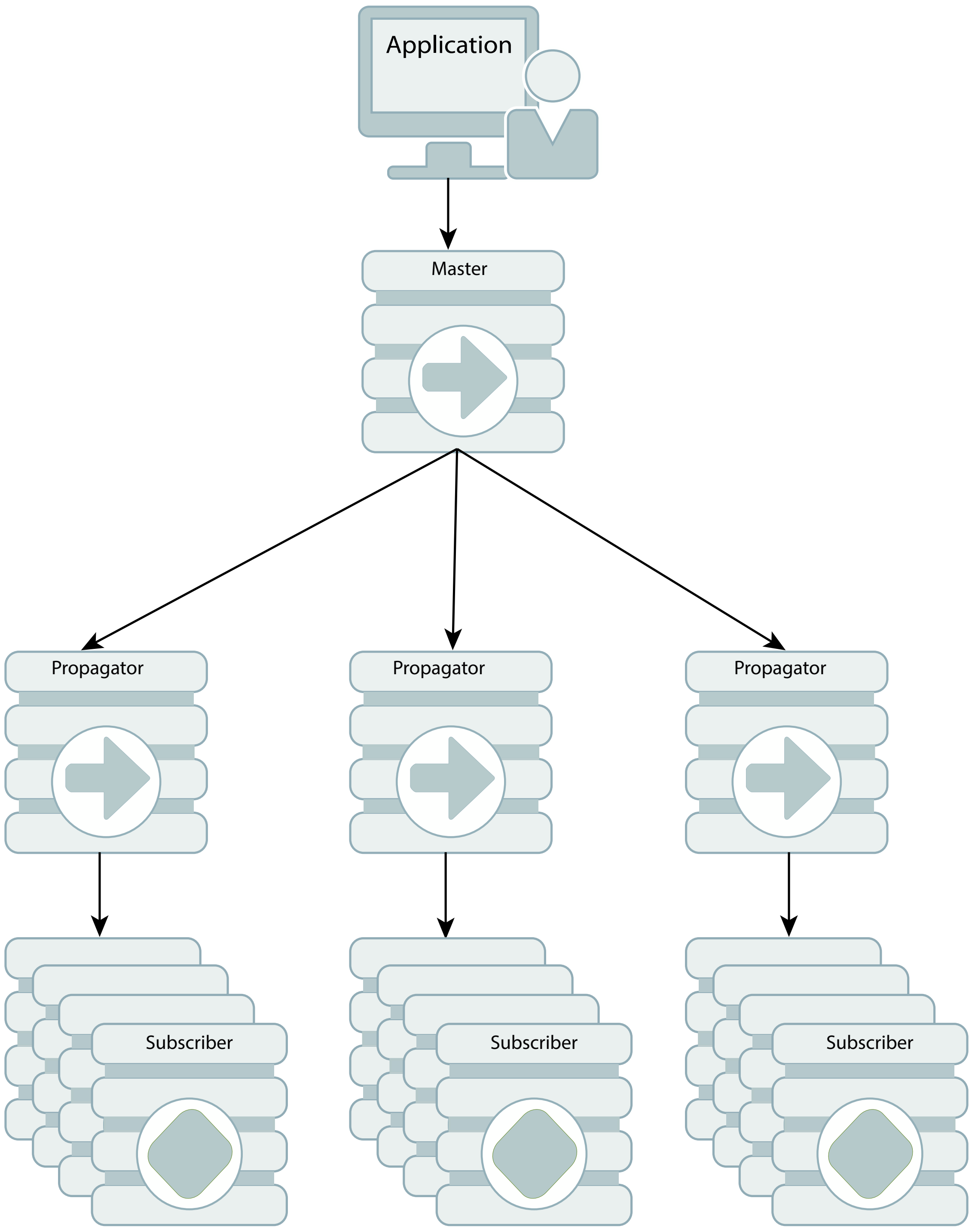
For optimum performance, consider the configuration shown in Figure 1-10, where the master replicates to a single propagator over the network connection. The propagator in turn forwards the updates to each subscriber on its local area network.

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.

Configuring a Large Number of Subscribers

By default, a replication scheme can include up to 255 subscribers. A replication scheme with propagator databases can have up to 255 propagators, and each propagator can have up to 255 subscribers. An active standby pair replication scheme can include up to 254 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 the file system. 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.

Cache Groups and Replication

A cache group is a group of tables stored in a central Oracle database that are cached in local cache tables on TimesTen. Cache groups can be replicated between TimesTen databases. You can achieve high availability by using an active standby pair to replicate AWT 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

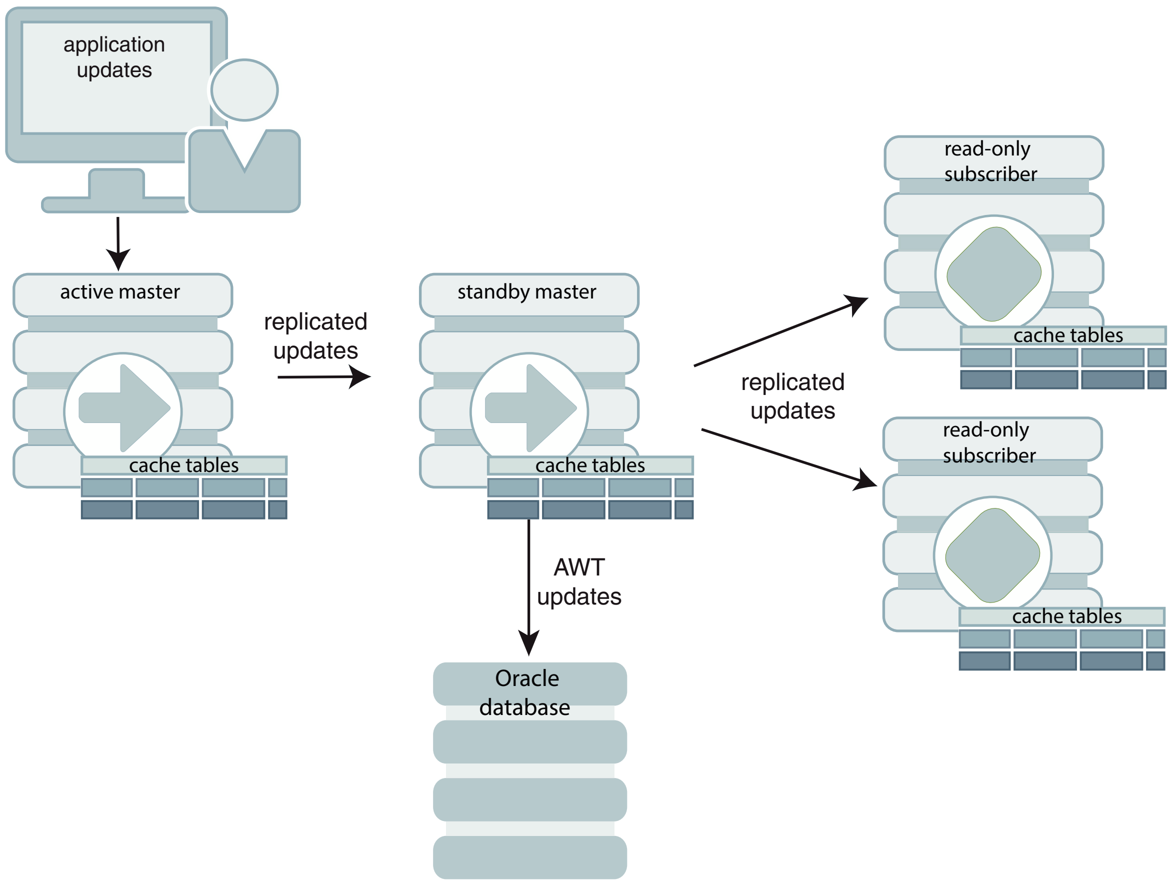
See Administering an Active Standby Pair with Cache Groups.

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 database 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 Classic 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 master becomes the new active master. TimesTen Classic automatically reconfigures the AWT cache group so that it can be updated directly by the application and continue to propagate the updates to the Oracle database asynchronously.

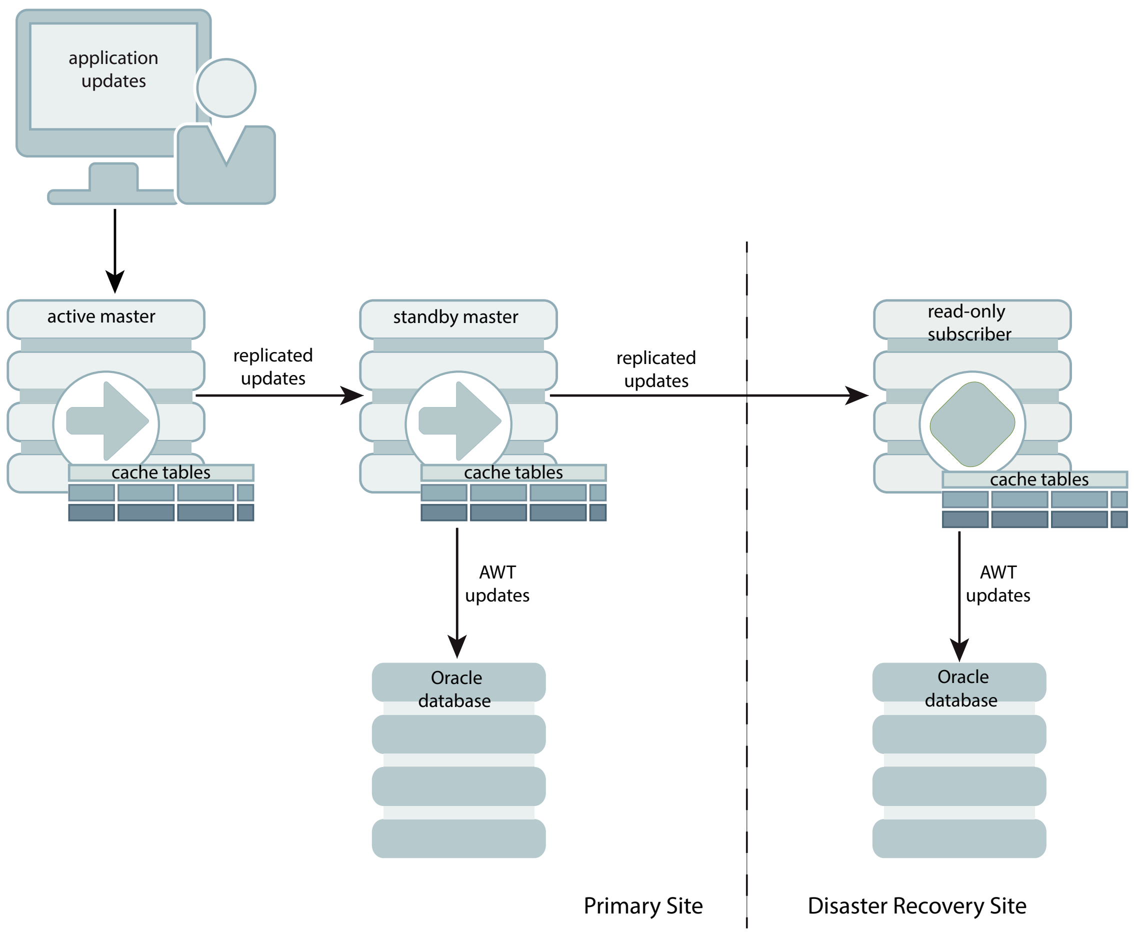
See Setting Up an Active Standby Pair with an AWT Cache Group.

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



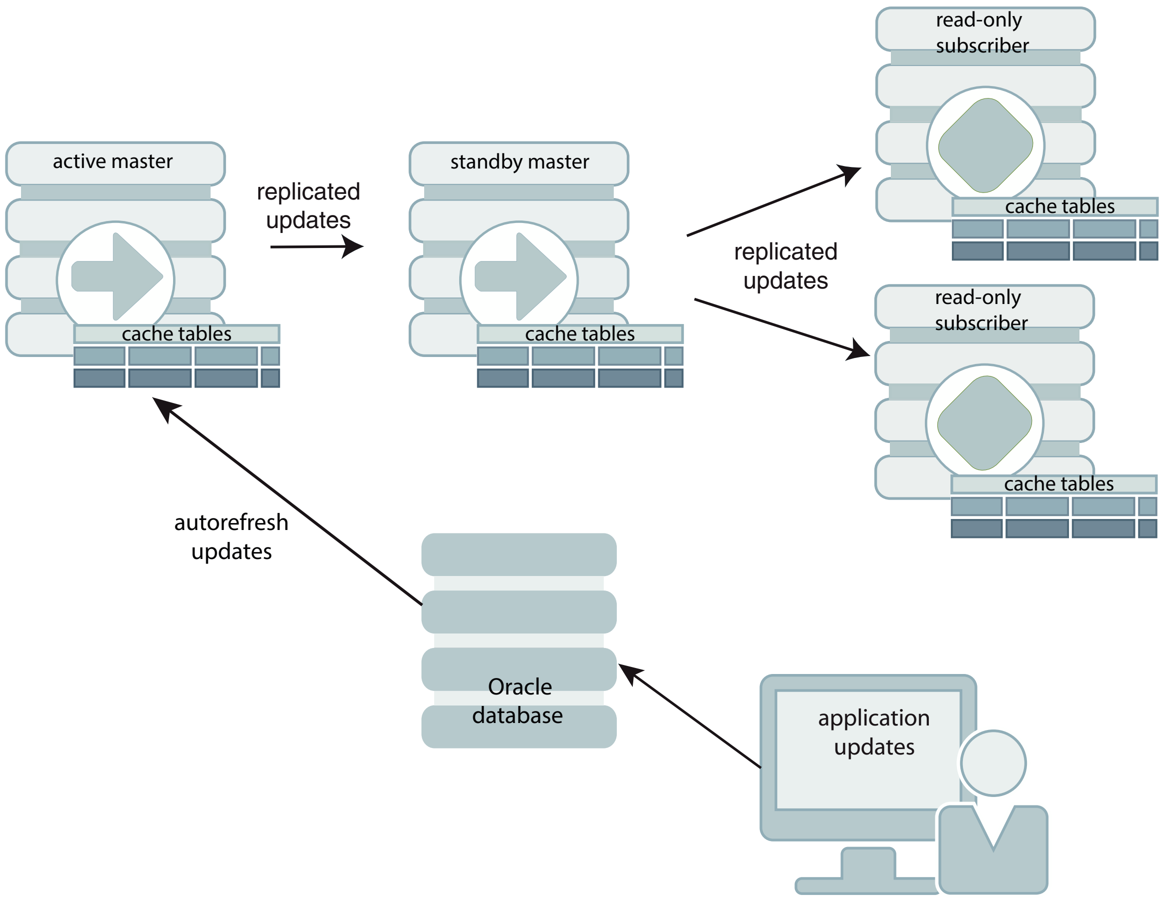
The standby database sends updates to cache group tables on the read-only subscriber. This special subscriber is located at a remote disaster recovery site and can propagate updates to a second Oracle database, also located at the disaster recovery site. You can set up more than one disaster recovery site with read-only subscribers and Oracle databases. See Using a Disaster Recovery Subscriber in an Active Standby Pair.

Replicating a Read-Only Cache Group

A read-only cache group enforces caching behavior in which committed updates on the Oracle database tables are automatically refreshed to the corresponding cache tables on TimesTen.

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 Classic automatically reconfigures the standby to be autorefreshed when it takes over for the failed master database by setting the `AUTOREFRESH STATE` to `ON`. TimesTen Classic 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 enables 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. See Setting Up an Active Standby Pair with a Read-Only Cache Group.

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 Classic 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.

Foreign Keys and Replication

You may choose to replicate all or a subset of tables that have foreign key relationships with one another. However, the method for how to replicate the tables involved in the relationship differ according to the type of replication scheme.

See the following for details:

- Replicating Tables with Foreign Key Relationships in an Active Standby Pair
- Replicating Tables with Foreign Key Relationships in a Classic Replication Scheme

Aging and Replication

There are rules that apply to the interaction with replication when a table or cache group is configured with least recently used (LRU) or time-based aging.

- 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

These examples demonstrate how to configure and start up sample replication schemes.

- Configuring an Active Standby Pair with One Subscriber
- Configuring a Classic Replication Scheme with One Master and One Subscriber
- Starting and Stopping the Replication Agents

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

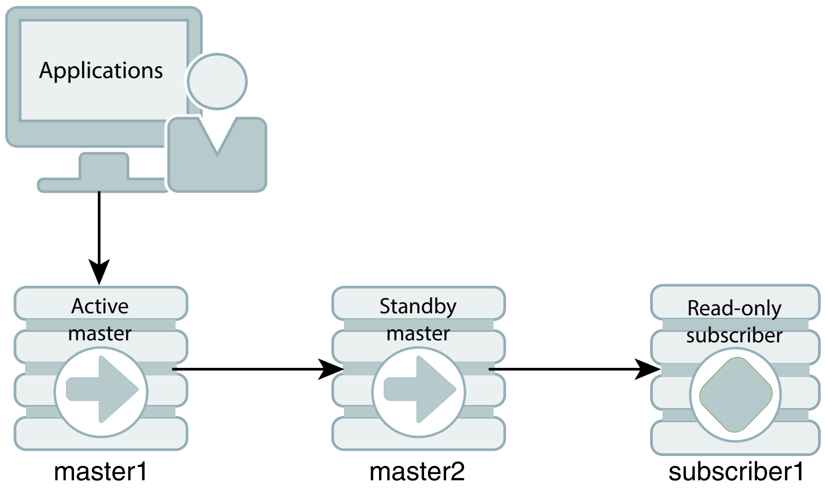
Configuring an Active Standby Pair with One Subscriber

You can create an active standby pair with a single 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



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*.

   Use a text editor to create the following `odbc.ini` file:

```
[master1]
DataStore=/tmp/master1
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8

[master2]
DataStore=/tmp/master2
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8

[subscriber1]
DataStore=/tmp/subscriber1
DatabaseCharacterSet=AL32UTF8
ConnectionCharacterSet=AL32UTF8
```

2. Optional: Enable TLS to encrypt communication between replication agents.

   You can enable secure TCP/IP network connections between replication agents (and utilities that communicate with the replication agents) by using Transport Layer Security (TLS), which requires mutual authentication to encrypt communication over connections. You can also generate certificates. See Transport Layer Security for TimesTen Replication in *Oracle TimesTen In-Memory Database Security Guide*.

   For encrypted databases, TLS must be enabled. The cipher suites and replication wallet settings between replication peers must match and be explicitly required to match (`replication_ssl_mandatory=1`).

3. Create a table in one of the master databases.

   a. 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=timesten;DataStore=/tmp/master1;
   DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
   (Default setting AutoCommit=1)
   Command>
   ```

   b. Create the `employees` table;

   ```
   Command> CREATE TABLE employees
       ( employee_id   NUMBER(6) PRIMARY KEY,
        first_name     VARCHAR2(20),
        last_name      VARCHAR2(25) NOT NULL,
        email          VARCHAR2(25) NOT NULL UNIQUE,
        phone_number   VARCHAR2(20),
        hire_date      DATE NOT NULL,
        job_id         VARCHAR2(10) NOT NULL,
        salary         NUMBER(8,2),
        commission_pct NUMBER(2,2),
        manager_id     NUMBER(6),
        department_id  NUMBER(4)
     ) ;
   ```

4. Define the active standby pair.

   The following defines the active standby pair on `master1`:

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

See Defining an Active Standby Pair Replication Scheme.

5. Start the replication agent on a master database.

   The following starts the replication agent on `master1`:

   ```
   Command> CALL ttRepStart;
   ```

   See Starting and Stopping the Replication Agents.

6. 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:

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

   Verify the state of `master1`:

   ```
   Command> CALL ttRepStateGet;
   < ACTIVE >
   1 row found.
   ```

7. Create a user on the active database.

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

   ```
   Command> CREATE USER ttuser IDENTIFIED BY ttuser;
   User created.
   Command> GRANT ADMIN TO ttuser;
   ```

8. Duplicate the active database to the standby database.

   Exit `ttIsql` and use the `ttRepAdmin` utility as the `ttuser` (the user created with the `ADMIN` privilege) 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 ttuser -pwd ttuser
   master2
   ```

9. Start the replication agent on the standby database.

   Use `ttIsql` to connect to the `master2` database 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=timesten;DataStore=/tmp/master2;
   DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
   (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 >
   1 row found.
   ```

10. Duplicate the standby database to the subscriber.

Exit `ttIsql` and use the `ttRepAdmin` utility as the `ttuser` (the user created with the `ADMIN` privilege) to duplicate the standby database to the subscriber database:

```
% ttRepAdmin -duplicate -from master2 -host server1 -uid ttuser -pwd ttuser
subscriber1
```

11. Start the replication agent on the subscriber.

    Use `ttIsql` to connect to `subscriber1` and start the replication agent. Verify the state of `subscriber1`. Starting the replication agent for the subscriber database automatically sets its state to `'IDLE'`. All subscriber databases have their state set to `'IDLE'`.

```
% 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=timesten;DataStore=/stmp/subscriber1;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
(Default setting AutoCommit=1)

Command> CALL ttRepStart;

Command> call ttRepStateGet;
< IDLE >
1 row found.
```

12. Insert data into the table on the active database.

    a. 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=timesten;DataStore=/tmp/master1;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
(Default setting AutoCommit=1)
Command>
```

    b. Insert a row into the `employees` table on `master1`.

```
Command> INSERT INTO employees VALUES
      ( 202,
        'Pat',
        'Fay',
        'PFAY',
        '603-123-7777',
        TO_DATE('17-AUG-1997', 'dd-MON-yyyy'),
        'MK_REP',
        6000,
        NULL,
        201,
        20
   );
1 row inserted.
Command> SELECT * FROM employees;
< 202, Pat, Fay, PFAY, 603-123-7777, 1997-08-17 00:00:00, MK_REP,
 6000, <NULL>, 201, 20 >
1 row found.
```

c. Verify that the insert is replicated to `master2` and `subscriber1`. Use `ttIsql` to connect to `master2`:

```
% 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=timesten;DataStore=/tmp/master2;
DatabaseCharacterSet=AL32UTF8;ConnectionCharacterSet=AL32UTF8;
(Default setting AutoCommit=1)
```

d. Verify that the data is replicated to `master2`:

```
Command> SELECT * FROM employees;
< 202, Pat, Fay, PFAY, 603-123-7777, 1997-08-17 00:00:00, MK_REP,
6000, <NULL>, 201, 20 >
1 row found.
```

e. Perform the same step on `subscriber1` to verify the data is replicated to the subscriber.

13. Drop the active standby pair and the table.

a. Connect to each database using `ttIsql` and stop the replication agents on each database:

```
Command> CALL ttRepStop;
```

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

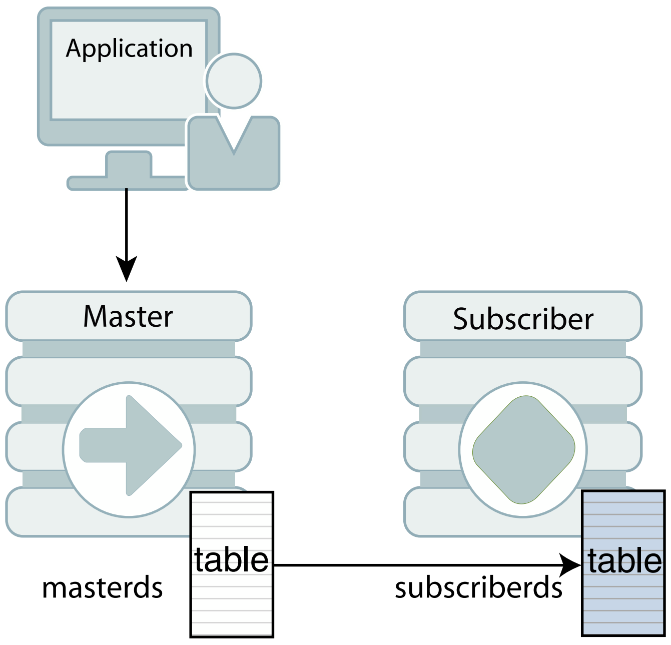
```
Command> DROP ACTIVE STANDBY PAIR;
Command> DROP TABLE employees;
```

Configuring a Classic Replication Scheme with One Master and One Subscriber

You can configure a classic replication scheme that replicates the contents of a single table in a master database to a table in a subscriber database.

This section describes how to configure a classic 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 Classic Replication Scheme



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*.

   Use a text editor to create the following `odbc.ini` file:

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

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

2. Optional: Enable TLS to encrypt communication between replication agents.

   You can enable secure TCP/IP network connections between replication agents (and utilities that communicate with the replication agents) by using Transport Layer Security (TLS), which requires mutual authentication to encrypt communication over connections. You can also generate certificates. See Transport Layer Security for TimesTen Replication in *Oracle TimesTen In-Memory Database Security Guide*.

   For encrypted databases, TLS must be enabled. The cipher suites and replication wallet settings between replication peers must match and be explicitly required to match (`replication_ssl_mandatory=1`).

3. Create a table and classic replication scheme on the master database.

   a. 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=timesten;
DataStore=/tmp/masterds;DatabaseCharacterSet=AL32UTF8;
ConnectionCharacterSet=AL32UTF8;
(Default setting AutoCommit=1)
Command>
```

b. Create the `employees` table:

```
Command> CREATE TABLE employees
    ( employee_id   NUMBER(6) PRIMARY KEY,
     first_name     VARCHAR2(20),
     last_name      VARCHAR2(25) NOT NULL,
     email          VARCHAR2(25) NOT NULL UNIQUE,
     phone_number   VARCHAR2(20),
     hire_date      DATE NOT NULL,
     job_id         VARCHAR2(10) NOT NULL,
     salary         NUMBER(8,2),
     commission_pct NUMBER(2,2),
     manager_id     NUMBER(6),
     department_id  NUMBER(4)
  ) ;
```

c. Create a classic replication scheme called `repscheme` to replicate the `employees` table from `masterds` to `subscriberds`.

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

4. 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.

5. 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.
```

See Starting and Stopping the Replication Agents.

6. Insert data into the table on the master database.

   a. Use `ttIsql` to connect to the master database and insert some rows into the `employees` table:

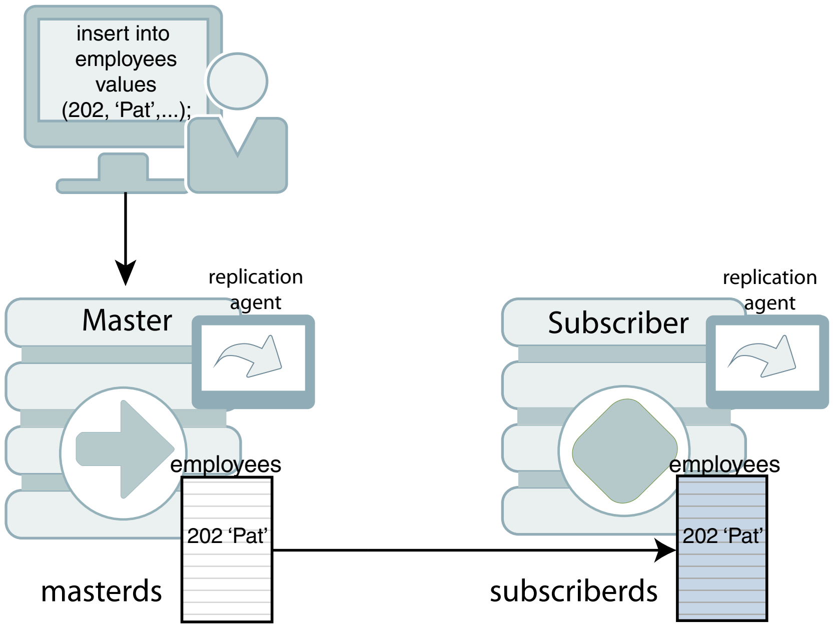
```
% ttIsql masterds
Command> INSERT INTO employees VALUES
      ( 202,
        'Pat',
        'Fay',
        'PFAY',
        '603-123-7777',
        TO_DATE('17-AUG-1997', 'dd-MON-yyyy'),
        'MK_REP',
        6000,
        NULL,
        201,
        20
   );
1 row inserted.
```

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

```
% ttIsql subscriberds
Command> SELECT * FROM employees;
< 202, Pat, Fay, PFAY, 603-123-7777, 1997-08-17 00:00:00, MK_REP,
6000, <NULL>, 201, 20 >
1 row 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



7. Drop the classic replication scheme and table.

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

   ```
   Command> CALL ttRepStop;
   ```

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

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

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.

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

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

To stop the replication agents, enter:

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

To start and stop the replication agent for the DSN named `masterDSN` from `ttIsql`, 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.

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*.

3

Defining an Active Standby Pair Replication Scheme

You can design a highly available system and define replication schemes.

To reduce the amount of bandwidth required for replication, see Compressing Replicated Traffic.

Overview of Master Database States

Overview of 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.

Restrictions on Active Standby Pairs

Keep in mind the restrictions when you are planning an active standby pair replication scheme.

- To ensure high availability, each active and standby master databases as well as all subscriber databases should be on different machines.
- The Linux or UNIX platforms for each host on which the master databases reside should have the same operating system kernel settings for shared memory and semaphores.
- For best replication performance a fast, stable network is best. The following can stall or impede replication progress.
  - Slow network: A slow network directly affects transaction rate of replication. Enabling compression could help. See Store Data Efficiently with Column-Based Compression of Tables in the *Oracle TimesTen In-Memory Database Operations Guide*.
  - Network outage: In the event of a network outage, replication operations stop and only resume when a connection is active between sender and receiver.
- The active and standby masters must have their clocks synchronized through NTP or other means. The clock skew between the active master and the standby master cannot exceed 250 milliseconds. When adjusting the system clocks on any nodes to be synchronized with each other, do not set any clock backward in time.
- For the initial setup, you create the standby database by duplicating the active database with the `ttRepAdmin -duplicate` utility or the `ttRepDuplicateEx` C function.
- `ALTER ACTIVE STANDBY PAIR` statements can run only on the active database. If `ALTER ACTIVE STANDBY PAIR` is run 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.
- 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.
- You can specify up to 254 subscribers,
- Replication from the standby database to the read-only subscribers occurs asynchronously.
- Write operations on replicated tables are not allowed on the standby database or the subscriber databases. However, operations on sequences and XLA bookmarks *are* allowed on the standby database and the subscriber databases. Read operations are also allowed.
- 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.
- 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 or Linux, create an `odbc.ini` file. See Configuring an Active Standby Pair with One Subscriber 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. For example, if the database path is *`directory/subdirectory/`*`foo.ds0`, then `foo` is the database name that you should use.

Table Requirements for Active Standby Pairs

Before you can create an active standby pair, you must create an object to be replicated.

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 restrictions:

- A primary key column cannot have a LOB data type.
- You cannot replicate tables with compressed columns.

Defining an Active Standby Pair Replication Scheme

Use the `CREATE ACTIVE STANDBY PAIR` SQL statement to create an active standby pair replication scheme. 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.

Note

See Configuring an Active Standby Pair with One Subscriber for an example. See CREATE ACTIVE STANDBY PAIR for the complete syntax in the *Oracle TimesTen In-Memory Database SQL Reference*.

Table 3-1 shows the components of the `CREATE ACTIVE STANDBY PAIR` statement that are used to create the active standby pair replication scheme. Each component is described with the identified 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
[*`ReturnServiceAttribute`*]	Using a Return Service
[`SUBSCRIBER` *`FullDatabaseName`* [,...]]	Identifying the Databases in the Active Standby Pair
[`STORE` *`FullDatabaseName`* [*`StoreAttribute`* [...]]]	Setting STORE Attributes
[*`NetworkOperation`* [...]]	Configuring Network Interfaces with the ROUTE Clause
`[{INCLUDE|EXCLUDE}` `{TABLE [[`*`Owner`*`.]`*`TableName`*`[,...]]|` `CACHE GROUP [[`*`Owner`*`.]`*`CacheGroupName`*`[,...]]|` `SEQUENCE [[`*`Owner`*`.]`*`SequenceName`*`[,...]]}` `[,...]]`	Including or Excluding Database Objects from Replication Replicating Cache Groups Within Active Standby Pairs

Identifying the Databases in the Active Standby Pair

The first component identifies the active master database, standby master database, and any subscriber databases. The first database name designates the active master database. The second database name designates the standby master database. Read-only subscriber databases are indicated by the `SUBSCRIBER` clause.

Use the full database name described in Defining the DSNs for the Databases.

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

The active master database and the standby master 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.

You can also specify the hosts where the databases reside by using an IP address or a literal host name surrounded by double quotes. 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 ON "host1",
   repdb2 ON "host2";
```

Using a Return Service for an Active Standby Pair

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 Using a Return Service.

Setting STORE Attributes for an Active Standby Pair

The `STORE` attributes clause in either the `CREATE ACTIVE STANDBY PAIR` or `ALTER ACTIVE STANDBY PAIR` statements are used to set optional behavior for return services, compression, timeouts, durable commit behavior, and table definition checking.

See CREATE ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference* for a full description of `STORE` attributes.

Note

If you are using `ALTER ACTIVE STANDBY PAIR` to change any of the `STORE` attributes, you must follow the steps described in Making Other Changes to an Active Standby Pair.

See Setting STORE Attributes for more details on how to use and configure the `STORE` attributes for an active standby pair.

Configuring Network Operations for an Active Standby Pair

If a replication host has more than one network interface, you may want to configure replication to use an interface other than the default interface.

See Configuring Network Interfaces with the ROUTE Clause.

Using Automatic Client Failover for an Active Standby Pair

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

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. See 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 are 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.

Replicating Tables with Foreign Key Relationships in an Active Standby Pair

With the active standby pair replication scheme, you may choose to replicate all or a subset of tables that have foreign key relationships with one another.

You can create the tables and the foreign key relationship on the active master either before or after the active standby pair replication scheme is created.

- Before creation of active standby pair: You can create the tables and the foreign key relationship on the active master before the active standby pair replication scheme is created. Then, create the active standby pair replication scheme.
- After creation of active standby pair: You can create the tables and the foreign key relationship on the active master after the active standby pair replication scheme is created. In order for the tables to be automatically replicated to the standby master and added to the replication scheme, you must be using the default mode where `DDLReplicationLevel` is set to 2 or larger and `DDLReplicationAction='INCLUDE'`. See Controlling Replication of Objects to All Databases in an Active Standby Pair.

If a child table with a foreign key defines `ON DELETE CASCADE`, then you must replicate any other table with a foreign key relationship to the child table. This requirement prevents foreign key conflicts from occurring on the standby master tables when a cascade deletion occurs on the active master database.

TimesTen Classic 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.

Replicating Cache Groups in an Active Standby Pair

With the active standby pair replication scheme, you must replicate all AWT cache groups, but you can choose to replicate any read-only cache groups.

You can create the cache groups on the active master either before or after the active standby pair replication scheme is created. See Replicating Cache Groups Within Active Standby Pairs.

Materialized Views in an Active Standby Pair

When you replicate a database containing a materialized or non-materialized 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 Classic automatically updates the corresponding view. However, TimesTen Classic 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.

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.

Duplicating a Database

When you set up a replication scheme or administer a recovery, a common task is to duplicate a database. Use the `-duplicate` option of the `ttRepAdmin` 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.
- You must use the `-keepCG` option when duplicating an active master database that has cache groups and provide the cache administration username and password.
- The target DSN cannot include client/server attributes.

Note

Enable the `Preallocate` attribute to ensure that there is sufficient space for the new database. If enabled, then TimesTen preallocates space on the file system for the database checkpoint files for the new database during the duplicate. See Preallocate in the *Oracle TimesTen In-Memory Database Reference*.

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

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

Command> GRANT ADMIN TO ttuser;
```

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

See ttRepAdmin in *Oracle TimesTen In-Memory Database Reference*and ttRepDuplicateEx in *Oracle TimesTen In-Memory Database C Developer's Guide*.

The following sections describe options when duplicating:

- Duplicating over a Specific Network Interface
- Duplicating with Cache Groups

Duplicating over a Specific Network Interface

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 Classic chooses them.

Duplicating with Cache Groups

If you are duplicating an active database that has defined cache groups, use the `-keepCG` option.

You must also specify the cache administration user name 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 name is `orauser` and the password is `orapwd`, duplicate database `dsn1` on `host1`:

```
ttRepAdmin -duplicate -from dsn1 -host host1 -keepCG
 -connStr "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. The `ttRepAdmin` utility 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 -connStr "DSN=sub1;UID=;PWD="
```

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

If you cannot access the Oracle database (either the Oracle database is down or you cannot connect to it) while performing a duplicate for a replication scheme with AWT cache groups or cache groups with incremental autorefresh, then the `ttRepAdmin -duplicate` command cannot update the metadata on the Oracle database (that cache uses to manage AWT cache groups and cache groups with autorefresh) after AWT cache groups or cache groups with incremental autorefresh are duplicated. In this case, use one of the following options to perform the duplicate:

- If you are using `ttRepAdmin -duplicate` to recover either a failed active or standby master where all AWT cache groups or cache groups with incremental autorefresh are included in the active standby pair replication scheme, then use the `-keepCG -recoveringNode` options. When this option is used, changes that occur during the duplicate operation are tracked and so may not need to initiate a full autorefresh.

- Otherwise, use the `-keepCG -deferCacheUpdate` options. This option may initiate a full autorefresh.

After completion of the duplicate operation with either `-keepCG -recoveringNode` options or `-keepCG -deferCacheUpdate` options, warning messages are posted informing you that while the duplicate operation was successful, updates to the Oracle database metadata are deferred until the cache and replication agents are started. Thus, once the duplicate operation is complete, start both the cache and replication agents on the new node. If there are cascading TimesTen node failures and intermittent connectivity problems with the Oracle database, then starting the cache and replication agents may initiate a full autorefresh.

Note

See Replicating Cache Groups Within Active Standby Pairs.

4

Defining Attributes and Options for a Replication Scheme

There are return service options, `STORE` attributes, and network operations that can be configured for both active standby pairs and classic replication (involving master and subscribers).

Any differences for one replication scheme over the other are detailed within each section.

Connection Attributes for Replicated Databases

Databases that replicate to each other must have the same `DatabaseCharacterSet` data store attribute. TimesTen Classic does not perform any character set conversion between replicated databases.

You must make sure that the underlying data type for each replicated column is the same on each node when you replicate between databases.

See Managing the Transaction Log on a Replicated Database for recommendations for managing the replication log files.

If you want to configure parallel replication, see Configuring Parallel Replication for information about setting the `ReplicationParallelism` and `ReplicationApplyOrdering` data store attributes.

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 applying thread, 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.

These threads act in parallel to replicate and apply transactional changes to nodes in a replication scheme. By default, parallel replication enforces transactional dependencies and applies changes in commit order; however, you can disable enforcement of the commit order.

Note

If you enable parallel replication, you cannot run both DDL and DML statements in the same transaction.

Parallel replication options:

- Automatic parallel replication: Parallel replication over multiple threads that automatically enforces transactional dependencies and all changes applied in commit order. This is the default.
- Automatic parallel replication with disabled commit dependencies: Parallel replication over multiple threads that automatically enforces transactional dependencies, but does not enforce transactions to be committed in the same order on the subscriber database as on the master database.

These options are configured with the `ReplicationApplyOrdering` and `ReplicationParallelism` data store creation attributes, which must be set when the database is created.

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.

See Upgrades When Using Parallel Replication in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

The following sections describe the options for parallel replication:

- Configuring Automatic Parallel Replication
- Configuring Automatic Parallel Replication with Disabled Commit Dependencies

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 either a classic or an active standby pair replication scheme.

Automatic parallel replication enforces transactional dependencies and applies changes in commit order.

Enable automatic parallel replication by setting these data store attributes at database creation time:

- Set `ReplicationApplyOrdering=0`, which is also the default.
- Set `ReplicationParallelism` to a number from 2 to 32. This number indicates the number of transmitter threads on the source database and the number of receiver threads on the

target database. However, if you are using single-threaded replication, set `ReplicationParallelism` to 1, which is the default.

The `LogBufParallelism` and `ReplicationParallelism` connection attributes are related. `LogBufParallelism` specifies the number of strands that are mapped to the threads that are specified by `ReplicationParallelism`. For example, if `LogBufParallelism` = 4 and `ReplicationParallelism` = 4, then one strand is mapped to one thread. If `LogBufParallelism` = 8 and `ReplicationParallelism` = 4, then two strands are mapped to one thread.

Thus, if `ReplicationParallelism` is greater than 1, the `LogBufParallelism` connection attribute must be equal to or greater than the value of `ReplicationParallelism`. The `ReplicationParallelism` connection attribute cannot exceed the value of `LogBufParallelism`. In order for the number of strands to be equally distributed across the number of threads, you may want to make `LogBufParallelism` a multiple of the number of threads specified in `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 database tables. See Improving AWT Throughput with Parallel Propagation to the Oracle Database in *Oracle TimesTen In-Memory Database Cache Guide*.

See ReplicationParallelism, ReplicationApplyOrdering, and LogBufParallelism in the *Oracle TimesTen In-Memory Database Reference*.

Configuring Automatic Parallel Replication with Disabled Commit Dependencies

In order to enforce transactional dependencies and ensure that changes are applied in commit order, automatic parallel replication normally tracks begin and commit dependencies.

- Begin dependencies: Operations that force one transaction to run before another, such as an insert of a row followed by a delete of that same row.
- Commit dependencies: Order in which transactions are committed so that they are committed on a subscriber in the same order as on the master database.

While parallel replication improves performance by utilizing multiple threads, you can further increase throughput performance when using automatic parallel replication if the transactions do not require enforcement of commit dependencies. That is, 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 relax the enforcement of the commit dependencies and still maintain transactional correctness. For example, if separate transactions are working on separate tables, you do not need to enforce commit dependencies.

By relaxing the requirement for tracking commit dependencies, you can improve the performance for automatic parallel replication. When you do not enforce commit dependencies, all non-DDL transactions may commit on the subscribers in a different order than how they were originally run on the master. Begin dependencies are always enforced to prevent apply order anomalies.

You can only disable commit dependencies for automatic parallel replication for an active standby pair that uses asynchronous replication and does not contain cache groups. You can set the following data store attributes at database creation time:

- Set the `ReplicationApplyOrdering=2` before you create the TimesTen database.

- Set `ReplicationParallelism` to a number from 2 to 32. This number indicates the number of transmitter threads on the source database and the number of receiver threads on the target database. However, if you are using single-threaded replication, set `ReplicationParallelism` to 1, which is the default.

  The `LogBufParallelism` and `ReplicationParallelism` connection attributes are related. `LogBufParallelism` specifies the number of strands that are mapped to the threads that are specified by `ReplicationParallelism`. For example, if `LogBufParallelism` = 4 and `ReplicationParallelism` = 4, then one strand is mapped to one thread. If `LogBufParallelism` = 8 and `ReplicationParallelism` = 4, then two strands are mapped to one thread.

  Thus, if `ReplicationParallelism` is greater than 1, the `LogBufParallelism` connection attribute must be equal to or greater than the value of `ReplicationParallelism`. The `ReplicationParallelism` connection attribute cannot exceed the value of `LogBufParallelism`. In order for the number of strands to be equally distributed across the number of threads, you may want to make `LogBufParallelism` a multiple of the number of threads specified in `ReplicationParallelism`.

However, while your performance improves, this option requires the use of additional space in the temporary region of 16 MB plus an additional 16 MB for each configured replication track (`ReplicationParallelism` connection attribute setting). For example, if the `ReplicationParallism` connection attribute is set to 10, then this feature requires 16 MB + 160 MB = 176 MB of additional temporary region space.

When you use automatic parallel replication and disable commit dependencies, some of the tracks may move ahead of the others. This is known as a drift between the parallel replication tracks. You can limit the amount of drift between the replication tracks by setting the `ParReplMaxDrift` configuration parameter within the `ttDbConfig` built-in procedure.

```
Call ttDbConfig("ParReplMaxDrift", "30");
```

The example sets the allowed drift to 30 seconds between the replication tracks, after which TimesTen Classic causes all replication tracks to catch up to each other. See ttDBConfig in the *Oracle TimesTen In-Memory Database Reference*.

Specifying Replication Tracks Within an Automatic Parallel Replication Environment

In general, automatic parallel replication decides over which thread (track) to replicate each transaction from the master to the subscriber. Thus, you do not have to manually decide how to divide work across different tracks.

However, with dependent transactions, you may achieve better performance by manually assigning transactions to the same track.

Thus, the application has the option to specify which track a transaction belongs to when the transaction starts on the source database with either the `ReplicationTrack` connection attribute or the `ALTER SESSION SET REPLICATION_TRACK` statement. After which, all transactions for this connection use this track. The transactions for this 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. If you decide to specify the track for a transaction, then make sure that you distribute the workload evenly across the tracks.

Updates that should be applied in order on the receiving side should use the same track. You can spread operations on a table across separate tracks based on key values. For example, if you have a telecommunications billing application, you can use hash of the account number to set the track and send all transactions for each account on a separate track.

TimesTen Classic still computes and enforces dependencies to make sure that dependent transactions are applied in the correct order on the receiving side.

The application assigns transactions to tracks by one of these methods:

- Set the `ReplicationTrack` general connection attribute to a nonzero 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.
- Use the `setReplicationTrack()` method of the `TimesTenConnection` JDBC class.

Use the `ttConfiguration` built-in procedure to return the replication track number for the current connection. Select from the `SYS.GV$LOG_HOLDS` or `SYS.V$LOG_HOLDS` system views or call the `ttLogHolds` built-in procedure to verify that multiple tracks are being used.

Managing the Transaction Log on a Replicated Database

You can manage the transaction log on replicated databases.

This section includes these topics:

- About Log Buffer Flushing
- About Transaction Log Growth on a Master Database
- Setting Connection Attributes for Logging

About Log Buffer Flushing

A dedicated subdaemon thread writes the contents of the log buffer to the file system periodically. These write operations 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 write operations to the file system are synchronous write operations and the data is durably written to the file system before the write call returns. If the database is configured with `LogFlushMethod=1`, then the write operations are buffered unless there is a specific request from an application for synchronous write operations.

In addition to the periodic write operations, an application can also trigger the subdaemon thread to write the buffer contents to the file system. The following are cases where the application triggers a synchronous write operation to the file system:

- 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 runs a durable commit, whether the master database is configured with `TRANSMIT DURABLE` or not.

Transactions are also written to the file system 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 the file system 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.

The unneeded transaction log files are purged 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. Select from the `SYS.V$LOG_HOLDS` system view or call the `ttLogHolds` built-in procedure to get information about replication log holds.

Note

See Monitoring Accumulation of Transaction Log Files in *Oracle TimesTen In-Memory Database Operations Guide*.

See Monitor Replication from the Replication Log Holds in this book or SYS.V$LOG_HOLDS in the *Oracle TimesTen In-Memory Database System Tables and Views Reference* or ttLogHolds in the *Oracle TimesTen In-Memory Database Reference*

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 when it becomes full.

The minimum size for `LogBufMB` is 8 times the value of `LogBufParallelism`.

You need to establish enough space for the transaction log files. There are two settings that control the amount of 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. 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 the file system 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 Transaction Log Failure Threshold.

Because transactions are logged to the file system, 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 the file system. 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.

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 Managing Database Failover and Recovery.

See Connection Attributes in the *Oracle TimesTen In-Memory Database Reference*.

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 databases in your replication scheme.

Note

This section assumes you understand return services. For an overview on return services, see Copying Updates Between 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 standby or subscriber defined in replication scheme with the `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR`, `CREATE REPLICATION`, or `ALTER REPLICATION` statements. The default is the `NO RETURN` service, which is asynchronous replication and the best performance option.

Note

You can use the `ttRepXactStatus` procedure to check on the status of a return receipt or return twosafe transaction. See Check the Status of Return Service Transactions.

The following sections describe the return services that can be used for your replication scheme:

- RETURN RECEIPT
- RETURN RECEIPT BY REQUEST
- RETURN TWOSAFE
- RETURN TWOSAFE BY REQUEST
- NO RETURN
- Specifying a Different Return Service for Each Subscriber in a Classic Replication Scheme
- Setting the Return Service Timeout Period
- Disabling Return Service Blocking Manually
- Establishing Return Service Failure and Recovery Policies

RETURN RECEIPT

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

- In an active standby pair, 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.
- In a classic replication scheme, you can specify the `RETURN RECEIPT` clause to enable the return receipt service for the subscriber database. With return receipt enabled, when your 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.

Note

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.

If the standby or subscriber 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.

The following example defines return receipt for an active standby pair. This example creates an active standby pair where `master1` is the active database, `master2` is the standby database. The standby database is enabled with the return receipt service.

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  RETURN RECEIPT;
```

The following example defines return receipt for a classic replication scheme. 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:

Note

For more examples of classic replication schemes that use return receipt services, see Multiple Subscriber Classic Replication Schemes.

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

RETURN RECEIPT BY REQUEST

`RETURN RECEIPT` enables notification of receipt for all transactions.

You can use the `RETURN RECEIPT BY REQUEST` clause to enable an acknowledgement 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 on the active or master database 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 or subscriber 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.

The following example defines return receipt by request for an active standby pair. This example creates an active standby pair where `master1` is the active database and `master2` is the standby database. The standby database is enabled with the return receipt service.

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  RETURN RECEIPT BY REQUEST;
```

This example defines return receipt by request for a classic replication scheme. 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
```

You can use `ttRepSyncSet` to request return services. Before committing a transaction that requires an acknowledgement return receipt, call `ttRepSyncSet`. The following example sets the request for a return receipt with the first column set to `0x01` with a timeout value of 45 seconds in column two.

```
Command> autocommit off;
Command> CALL ttRepSyncSet(0x01, 45, 1);
```

You can use `ttRepSyncGet` to check if a return service is enabled and obtain the timeout value. The following demonstrates that the values that were previously set with the `ttRepSyncSet` built-in procedure.

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

See ttRepSyncSet and ttRepSyncGet in the *Oracle TimesTen In-Memory Database Reference*.

RETURN TWOSAFE

TimesTen Classic 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 or 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.

Note

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

To enable the return twosafe service for the subscriber, specify the `RETURN TWOSAFE` attribute in the `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR`, `CREATE REPLICATION`, or `ALTER REPLICATION` statements.

- When using an active standby pair, 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.
- When using a classic replication scheme, 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.

  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 standby or subscriber 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.

The following example defines return twosafe service for an active standby pair. This example creates an active standby pair where `master1` is the active database, `master2` is the standby database. The standby database is enabled with the return twosafe service.

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  RETURN TWOSAFE;
```

The following example defines return twosafe service for a classic replication scheme. 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;
```

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.

If you specify `RETURN TWOSAFE BY REQUEST` for a standby or subscriber database, you must use the `ttRepSyncSet` built-in procedure on the active or master database 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).

When you use the `ALTER TABLE` statement to alter a replicated table that is part of a `RETURN TWOSAFE BY REQUEST` transaction, it ends up not running as a part of the `TWOSAFE BY REQUEST` transaction. Instead, the `ALTER TABLE` operation succeeds because a commit is performed before the `ALTER TABLE` operation, resulting in the `ALTER TABLE` operation running in a new transaction which is not part of the `RETURN TWOSAFE BY REQUEST` transaction.

Note

See Setting the Return Service Timeout Period.

If the standby or 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.

When using an active standby pair, 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.

The following example defines return twosafe by request service for an active standby pair. This example creates an active standby pair where `master1` is the active database, `master2` is the standby database. The standby database is enabled with the return twosafe by request service.

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  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);
```

You can use the `ttRepSyncGet` built-in procedure to check if a return service is enabled and obtain the timeout value.

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

This example defines return twosafe by request for a classic replication scheme. 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);
```

You can use the `ttRepSyncGet` built-in procedure to check if a return service is enabled and obtain the timeout value.

```
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 services, depending on which one you have enabled.

`NO RETURN` is the default condition. This attribute is typically used only when altering a replication scheme to remove a previously defined return service in the `ALTER ACTIVE STANDBY PAIR` or `ALTER REPLICATION` statements.

Specifying a Different Return Service for Each Subscriber in a Classic Replication Scheme

In a classic replication scheme, you can specify a different return service for table elements and database elements for the subscribers listed in each `SUBSCRIBER` clause in a `CREATE REPLICATION` or `ALTER REPLICATION` statement.

The following example shows separate `SUBSCRIBER` clauses that can define different return service attributes for each subscriber: *`SubDatabase1`* and *`SubDatabase2`*.

```
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. The following example shows the use of a single `SUBSCRIBER` clause that defines the same return service attributes for both *SubDatabase1* and *SubDatabase2*.

```
CREATE REPLICATION Owner.SchemeName
  ELEMENT ElementNameElementType
    MASTER DatabaseName ON "HostName"
    SUBSCRIBER SubDatabase1 ON "HostName",
               SubDatabase2 ON "HostName"
               ReturnServiceAttribute;
```

Setting the Return Service Timeout Period

A timeout can occur in a replication scheme configured with one of the return services.

See Using a Return Service.

- In an active standby pair replication scheme, a timeout occurs if the standby database is unable to send an acknowledgement back to the active database 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.

- In a classic replication scheme, 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 replication state could be set 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.

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

The default return service timeout period is 10 seconds. You can specify a different return service timeout period by either:

- Specifying the `RETURN WAIT TIME` in the `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR`, `CREATE REPLICATION`, or `ALTER REPLICATION` statements.

  The `RETURN WAIT TIME` attribute specifies the number of seconds to wait for a return service acknowledgement. A value of 0 means that there is no waiting.

  The following example alters an active database (`master1`) of an active standby pair to set a return service wait time of 25 seconds:

  ```
  Command> ALTER ACTIVE STANDBY PAIR
    ALTER STORE master1 SET RETURN WAIT TIME 25;
  ```

- Specifying a different return service timeout period programmatically by calling the `ttRepSyncSet` built-in procedure on either the active database (in an active standby pair) or the master database (in a classic replication scheme) with a new timeout value for the `returnWait` parameter.

  The following example demonstrates how to set the return service wait time to 25 seconds using `ttRepSyncSet`:

  ```
  Command> CALL ttRepSyncSet (0x01, 25, 1);
  ```

Once the timeout is set, the timeout period applies to all subsequent return service transactions until you either reset the timeout period or terminate the application session. For a classic replication scheme, the timeout setting applies to all return services for all subscribers.

Note

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 Establishing Return Service Failure and Recovery Policies.

This example sets the timeout period for both databases included in a bidirectional classic 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;
```

This example shows how to reset 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);
```

Disabling Return Service Blocking Manually

You may want to react 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.

> **Note**
>
> One response to a timeout is to disable the return service. You can determine if the return service is enabled or disabled with either the `ttRepSyncSubscriberStatus` built-in procedure. See Determine If Return Service Is Disabled.

When using the return receipt service, you can manually respond by:

- Using the `ALTER ACTIVE STANDBY PAIR` or `ALTER REPLICATION` statements to disable return receipt blocking. 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.

  The following example uses the `ALTER ACTIVE STANDBY PAIR` statement to disable return receipt after 10 failures:

  ```
  Command> ALTER ACTIVE STANDBY PAIR
    ALTER STORE master1 SET DISABLE RETURN ALL 10;
  ```

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

Establishing Return Service Failure and 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`, `ALTER ACTIVE STANDBY PAIR`, `CREATE REPLICATION`, or `ALTER REPLICATION` statements 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. Except for DURABLE COMMIT, 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 two safe service continues to be enabled or is disabled when replication is paused or stopped.

- For an active standby pair, replication is considered stopped when either:
  – The active replication agent has failed or is explicitly stopped (for example, by `ttAdmin -repStop` *`active`*).

  - A failed standby master that has exceeded the specified `FAILTHRESHOLD` value stops replication. Note that even though replication is stopped when the standby master fails, TimesTen replicates directly from the active master to any subscribers, bypassing the standby master. All missing updates are propagated to the standby master if it recovers.
- In a classic replication scheme, replication is considered stopped when either:
  - The master replication agent is explicitly stopped (for example, by `ttAdmin -repStop` *`master`*).
  - The replication state of the subscriber database is set to `pause` or `stop` (for example, by `ttRepAdmin -state pause` *`subscriber`*).
  - A failed subscriber that has exceeded the specified `FAILTHRESHOLD` value stops replication.

Note

A standby or subscriber database may become unavailable for a period of time that exceeds the timeout period specified by `RETURN WAIT TIME`, yet may 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.

You can enable or disable the return service when replication is stopped with the following clause:

- `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` enables the return service to continue to be enabled when replication is stopped and is the default when using the `RETURN TWOSAFE` service.

The following example defines return services on when replication stopped for an active standby pair. This example creates an active standby pair with `RETURN TWOSAFE` return service and defines that the return service is to be disabled when replication is stopped (which is opposite of the default).

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  RETURN TWOSAFE
  STORE master2 RETURN SERVICES OFF WHEN REPLICATION STOPPED;
```

This example defines return services on when replication stopped for a classic replication scheme.

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, you could use the `ttRepAdmin -state pause` to set `subscriber1` to the `pause` state:

```
ttRepAdmin -receiver -name subscriber1 -state pause masterds
```

At this point, the application would continue to wait for return receipt acknowledgements from `subscriber1` until the replication state is reset to `start` and it receives the acknowledgment:

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

Note

See Set the Replication State of Subscribers. You should be cautious about setting the subscriber state to `stop`, as this not only stops the replication to this subscriber, but also discards all of the updates. If you did set the subscriber to the `stop` state, you would need to perform a duplicate to restore the subscriber.

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 or subscriber to acknowledge the replicated updates.

When return service blocking is disabled, the applications on the active or master database no longer blocks processing while waiting to receive acknowledgements from the standby or subscribers that they received or committed the replicated updates. Transactions are still replicated to the standby or subscriber, whether the return service is enabled or disabled. When the return service is disabled, the transactions are sent in asynchronous mode; the active or master database continues to listen for an acknowledgement of each batch of replicated updates from standby or subscriber databases.

Configure `DISABLE RETURN` as follows:

- For an active standby pair, specifying `SUBSCRIBER` is the same as specifying `ALL`. Both settings refer to the standby database.
- For a classic replication scheme, 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.

Note

You can use the `ttRepSyncSubscriberStatus` built-in procedure to determine whether the standby database or a particular subscriber has been disabled by the `DISABLE RETURN` failure policy.

The `DISABLE RETURN` failure policy is only enabled when the replication agent is running. If `DISABLE RETURN` is specified without `RESUME RETURN`, the return services remain off until the replication agent for the database has been restarted.

- For an active standby pair, you can cancel this failure policy by stopping the replication agent and specifying `DISABLE RETURN` with a zero value for *`NumFailures`*.
- For a classic replication scheme, 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`*.

  `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.

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.

This example shows how to set `DISABLE RETURN` for an active standby pair. Configure the `CREATE ACTIVE STANDBY PAIR` statement to replicate updates from the active database `master1` to the standby database `master2`. The `CREATE ACTIVE STANDBY PAIR` statement specifies the use of RETURN RECEIPT and `DISABLE RETURN ALL` with a *`NumFailures`* value of 5. The `RETURN WAIT TIME` is set to 30 seconds.

```
CREATE ACTIVE STANDBY PAIR
   master1,
   master2
   RETURN RECEIPT
   STORE master1
   DISABLE RETURN ALL 5
   RETURN WAIT TIME 30;
```

While the application is committing updates to the active database, the standby database (`master2`) experiences problems and fails to acknowledge a replicated transaction update. The application is blocked for 30 seconds after which it commits its next update to the active database `master1`. Over the course of the application session, this commit/timeout cycle repeats 4 more times until `DISABLE RETURN` disables return receipt blocking for `master2`.

The following example sets the `DISABLE RETURN SUBSCRIBER` for a classic replication scheme. Configure the `CREATE REPLICATION` statement to replicate updates from the `masterds` master database to the subscriber 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 for 30 seconds after which it commits its next update to the master database `masterds`. 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`.

RESUME RETURN

If `DISABLE RETURN` has disabled return service blocking, the `RESUME RETURN` attribute sets the policy for re-enabling the return service. You can establish a return service recovery policy by setting the `RESUME RETURN` attribute and specifying a resume latency value.

If return service blocking has been disabled for the standby or subscriber database and a latency time has been defined for `RESUME RETURN`, the following occurs:

- The applications on the active or master database no longer block processing while waiting to receive acknowledgements from the standby or subscribers. Transactions continue to be replicated to the standby or subscriber in asynchronous mode. The active or master databases continue to listen for an acknowledgement of each batch of replicated updates from standby or subscriber databases.
- If the return service blocking is disabled, `RESUME RETURN` evaluates the commit-to-acknowledge time for the last transaction to see if the latency is less than the latency limit configured by the `RESUME RETURN`. If the commit-to-acknowledge time latency is less than the latency limit set by `RESUME RETURN`, TimesTen re-enables the return receipt or return twosafe services.

Note

The commit-to-acknowledge time latency is the time elapsed between when the application issues a commit and when the active or master database receives acknowledgement from the standby or subscriber.

TimesTen evaluates the latency of the last acknowledged transaction before the current transaction is replicated to the standby or subscriber. The return service is re-enabled before the sending of the current transaction after evaluating the latency from the last transaction.

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` or `ALTER REPLICATION` statements to set `RESUME RETURN` to zero.

The following example sets `RESUME RETURN` for an active standby pair. If return receipt blocking has been disabled for `master2` and if `RESUME RETURN` is set to 8 milliseconds, then return receipt blocking is re-enabled for `master2` the instant the active receives an acknowledgement of the update from the standby, as long as the acknowledgement is received within the specified latency 8 milliseconds from when it was committed by the application on the active database.

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  RETURN RECEIPT
  STORE master1
  DISABLE RETURN ALL 5
  RESUME RETURN 8;
```

The following example sets `RESUME RETURN` for a classic replication scheme. 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 the master receives an acknowledgement of the update from the subscriber, as long as the acknowledgement is received within the specified latency 8 milliseconds from when it was committed by the application on the master database.

```
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;
```

DURABLE COMMIT

When `DURABLE COMMIT` is set to `ON`, it overrides the `DurableCommits` general connection attribute on the active or master database and forces durable commits for those transactions that have had return service blocking disabled.

In addition, when `DURABLE COMMIT` is set to `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 when the `ttRepStateSave` built-in procedure has marked the standby or subscriber database as failed.

For a classic replication scheme, `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.

Set `DURABLE COMMIT ON` for an active standby pair 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 the file system to provide redundancy.

```
Command> CREATE ACTIVE STANDBY PAIR
  master1,
  master2
  RETURN RECEIPT
  STORE master1
  DISABLE RETURN ALL 5
  DURABLE COMMIT ON
  RESUME RETURN 8;
```

Set `DURABLE COMMIT ON` within a classic replication scheme 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 the file system 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 you are using the return twosafe service, you can specify how the active or master replication agent responds to timeouts by setting `LOCAL COMMIT ACTION`. You can override this setting for specific transactions with the `localAction` parameter of the `ttRepSyncSet` built-in 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.

If the call returns with an error, you can use the `ttRepXactStatus` procedure described in Check the Status of Return Service Transactions 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 standby or subscriber, such as primary key lookup failure, you can roll back the transaction on the active or master database.

Setting STORE Attributes

The `STORE` attributes clause in the `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR`, `CREATE REPLICATION`, and `ALTER REPLICATION` statements are used to set optional behavior for return services, compression, timeouts, durable commit behavior, and table definition checking.

For a classic replication scheme, you can also define conflict reporting at the table level.

Note

See Using a Return Service in this book and CREATE ACTIVE STANDBY PAIR and CREATE REPLICATION in the *Oracle TimesTen In-Memory Database SQL Reference*.

When using classic replication schemes, the `FAILTHRESHOLD` and `TIMEOUT` attributes can be unique to a specific classic replication scheme definition. This means that these attribute settings can vary if you have applied different classic replication scheme definitions to your replicated databases. This is not true for any of the other attributes, which must be the same across all classic replication scheme definitions. For example, setting the `PORT` attribute for one classic replication scheme sets it for all classic replication schemes. For an example classic replication scheme that uses a `STORE` clause to set the `FAILTHRESHOLD` attribute, see Using a Return Service in a Classic Replication Scheme.

Note

If you are using `ALTER ACTIVE STANDBY PAIR` to change any of the `STORE` attributes, you must follow the steps described in Making Other Changes to an Active Standby Pair.

The following sections describe some of the `STORE` attributes:

- Column Definition Options for Replicated Tables
- Compressing Replicated Traffic
- Port Assignments
- Setting Wait Timeout for Response from Remote Replication Agents
- Setting the Transaction Log Failure Threshold
- Suspending or Resuming Classic Replication in Response to Conflicts

Column Definition Options for Replicated Tables

The definition for the columns of replicated tables participating in the replication scheme do not necessarily need to be identical.

- If the `TABLE DEFINITION CHECKING` value is set to `EXACT`, the column definitions must be identical on the active and standby databases. This attribute enables replication of tables that are identical in their physical structure.
- If the `TABLE DEFINITION CHECKING` value is set to `RELAXED` (the default), the column definitions of the replicated tables do not need to be identical. When using `RELAXED`, the replicated tables must have the same key definition, number of columns, column names, and column data types.

  Table definition checking occurs on the standby database. Setting this attribute to `RELAXED` for both active and standby databases has the same effect as setting it for only the standby database.

Note

See CREATE ACTIVE STANDBY PAIR or CREATE REPLICATION in the *Oracle TimesTen In-Memory Database SQL Reference*.

The `TABLE DEFINITION CHECKING RELAXED` attribute does not require that the physical structure of the table be identical on both master databases. For example, if tables have columns in a different order or have a different number of partitions, the data can still be replicated when using the `RELAXED` attribute. Thus, if you are altering your table by adding or dropping columns, you should use the `RELAXED` attribute. As noted in ALTER TABLE in the *Oracle TimesTen In-Memory Database SQL Reference*, adding columns when altering a table creates additional partitions. Dropping columns does not automatically free up the space. We recommend that any DML statement that alters the table should be run on the master and then replicated to any standby database and subscribers.

The `RELAXED` setting can result in slightly slower performance if it is compensating for a different physical structure. If the tables are identical in physical structure, then there is no performance impact. You can eliminate any performance issues (caused by a different physical structure, additional partitions, or extraneous space) by using the `ttMigrate -r -relaxedUpgrade` (only valid on databases where the table definition checking is set to `RELAXED`) to coalesce all additional partitions of a table into a single partition and eliminate extraneous space caused by dropped columns. If you perform this on all databases involved in the replication scheme, the resulting physical structure is identical resulting in the best performance potential.

Note

See Check Partition Counts for Tables in *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

For performance considerations of both the `EXACT` and `RELAXED` attributes for `TABLE DEFINITION CHECKING`, see Performance Considerations When Altering Tables That Are Replicated.

To ensure that table definition checking is set to `RELAXED`, stop the replication agent on the active or master database and then run an `ALTER ACTIVE STANDBY PAIR` or `ALTER REPLICATION` statement to set the table definition checking to `RELAXED`. Finally, use the `ttRepAdmin -duplicate` command to roll out these changes to the standby database and any subscribers. See ALTER ACTIVE STANDBY PAIR and ALTER REPLICATION in the *Oracle TimesTen In-Memory Database SQL Reference*.

The following sections provide examples for setting the table definition checking to relaxed:

- Setting Table Definition Checking to Relaxed for an Active Standby Pair
- Examples for Classic Replication Scheme with Table Definition Checking Set to Relaxed

Setting Table Definition Checking to Relaxed for an Active Standby Pair

This example demonstrates replicating tables that are identical in an active standby pair replication scheme. This example sets the `TABLE DEFINITION CHECKING` attribute to `EXACT`.

1. Create table `t1` in `master1` database:

   ```
   CREATE TABLE t1 (a INT PRIMARY KEY, b INT, c INT);
   ```

2. Create an active standby pair replication scheme. Set `TABLE DEFINITION CHECKING` to `EXACT` for the `master2` standby database.

   ```
   Command> CREATE ACTIVE STANDBY PAIR master1, master2
           STORE master2 TABLE DEFINITION CHECKING EXACT;
   ```

3. Perform the rest of the steps to duplicate the active database to the standby database, start the replication agents on both databases, and set the state of the active database (as described in Getting Started.

The following verifies that the table definition checking is enabled to exact.

1. Insert a row into `t1` on `master1`.

   ```
   Command> INSERT INTO t1 VALUES (4,5,6);
   1 row inserted.
   ```

2. Verify the results on `t1` on `master2`.

```
Command> SELECT * FROM t1;
< 4, 5, 6>
1 row found.
```

You can alter the table definition checking for the active standby pair replication scheme to be relaxed. First, stop the replication agent on the active database before altering the active database. The following alters the `dsn1` active database so the table definition checking is set to relaxed:

```
ALTER ACTIVE STANDBY PAIR
  ALTER STORE master1 SET TABLE DEFINITION CHECKING RELAXED;
```

After processing completes, use duplicate to roll out the changes to the standby database. Lastly, use duplicate to roll out the changes to any subscribers.

Examples for Classic Replication Scheme with Table Definition Checking Set to Relaxed

This provided example demonstrates replicating tables that are identical in a classic replication scheme. This example sets the `TABLE DEFINITION CHECKING` attribute to `EXACT`.

1. Create table `t1` in `dsn1` database:

   ```
   CREATE TABLE ttuser.t1 (a INT PRIMARY KEY, b INT, c INT);
   ```

2. Create `ttuser.t1` table in the `dsn2` database exactly the same as in the `dsn1` database.
3. Create replication scheme `ttuser.rep1`. Set `TABLE DEFINITION CHECKING` to `EXACT` for the subscriber, `dsn2`.

   ```
   CREATE REPLICATION ttuser.rep1
          ELEMENT e1 TABLE ttuser.t1
          MASTER dsn1
          SUBSCRIBER dsn2
          STORE dsn2 TABLE DEFINITION CHECKING EXACT;
   ```

4. 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.
   ```

5. Verify the results on `ttuser.t1` on `dsn2`.

   ```
   Command> SELECT * FROM ttuser.t1;
   < 4, 5, 6>
   1 row found.
   ```

The following example replicates tables with columns in different positions in a classic replication scheme. This example sets the `TABLE DEFINITION CHECKING` attribute to `RELAXED`.

1. Create table `t1` in `dsn1` database:

   ```
   CREATE TABLE ttuser.t1 (a INT PRIMARY KEY, b INT, c INT);
   ```

2. 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);
   ```

3. 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;
```

4. 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.
   ```

5. Verify the results on `ttuser.t1` on `dsn2`.

   ```
   Command> SELECT * FROM ttuser.t1;
   < 5, 6, 4 >
   1 row found.
   ```

The following example replicates tables with a different number of partitions in a classic replication scheme. 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.

1. Create table `ttuser.t3` on `dsn1` with two columns.

   ```
   CREATE TABLE ttuser.t3 (a INT PRIMARY KEY, b INT);
   ```

2. Create table `ttuser.t3` on `dsn2` with one column that is the primary key.

   ```
   CREATE TABLE ttuser.t3 (a INT PRIMARY KEY);
   ```

3. 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;
   ```

4. 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;
   ```

5. 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.
   ```

6. Verify the results in `ttuser.t3` on `dsn2`.

   ```
   Command> SELECT * FROM ttuser.t3;
   < 1, 2 >
   1 row found.
   ```

You can alter the table definition checking for a classic replication scheme to relaxed. First, stop the replication agent on the master database before altering the replication scheme on it. The following example alters the `dsn1` master database so the table definition checking is set to relaxed:

```
ALTER REPLICATION reppart
  ALTER STORE dsn1 SET TABLE DEFINITION CHECKING RELAXED;
```

After processing completes, use duplicate to roll out the changes to the standby master. Lastly, use duplicate to roll out the changes to any subscribers.

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`, `ALTER ACTIVE STANDBY PAIR`, `CREATE REPLICATION` or `ALTER REPLICATION` statements. 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.

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;
```

To compress the replicated traffic from both the active database `dsn1` and the standby database `dsn2`, 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;
```

You can compress replicated traffic from just one database `dsn1` in a classic replication scheme while leaving the replicated traffic from other databases (such as `dsn2`) uncompressed. Use the `CREATE REPLICATION` statement as follows:

```
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;
```

To compress the replicated traffic between both the `dsn1` and `dsn2` databases in a classic replicatoin scheme, 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

The `PORT` parameter for the `STORE` attribute of the `CREATE ACTIVE STANDBY PAIR` and `CREATE REPLICATION` statements set 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.
- In a classic replication scheme, the subscribers listen for updates from the master database. Setting the `PORT` attribute for one classic replication scheme sets it for all classic replication schemes.

Static port assignments are recommended. If no `PORT` attribute is specified, 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.

Note

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.

Example of assigning static ports for an active standby pair

```
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;
```

Example of assigning static ports for a classic replication scheme

```
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 Wait Timeout for Response from Remote Replication Agents

The `TIMEOUT` store attribute sets the maximum number of seconds that the replication agent waits for a response from any remote replication agents.

We recommend that the default timeout (120 seconds) is used if you have any large transactions. The replication agent scales the timeout based on the size of the transaction in order to accommodate any large transactions that could potentially cause a delayed response from the remote replication agent. Automatic scaling by the replication agent is disabled if the user sets the `TIMEOUT` to less than or equal to 60 seconds.

Note

If you experience repeated timeouts and the error log shows that multiple transmitter and receiver threads restart, then the transaction may be larger than can be scaled by the replication agent with the current timeout value. Continue to increase the timeout value until replication can progress for the transaction.

The following example creates an active standby pair whose master databases are `rep1` and `rep2`. There is one subscriber, `rep3`. The type of replication is `RETURN RECEIPT`. The statement also sets `PORT` and `TIMEOUT` attributes for the master databases. The `TIMEOUT` attribute is set to 80 seconds for both the active and standby masters.

```
CREATE ACTIVE STANDBY PAIR rep1, rep2 RETURN RECEIPT
   SUBSCRIBER rep3
 STORE rep1 PORT 21000 TIMEOUT 80
 STORE rep2 PORT 22000 TIMEOUT 80;
```

Setting the Transaction Log Failure Threshold

You can establish a threshold value that, when exceeded, sets an unavailable database to the `failed` state before the available transaction log space is exhausted.

- In an active standby pair, if the transaction log threshold is exceeded, sets an unavailable standby database or a read-only subscriber to the `failed` state before the available transaction log space is exhausted. Set the transaction log threshold by specifying the `STORE` clause with a `FAILTHRESHOLD` value in the `CREATE ACTIVE STANDBY PAIR` or `ALTER ACTIVE STANDBY PAIR` statements.

  If an active database sets the standby or read-only subscriber database to the `failed` state, it drops all of the data for the failed database from its transaction 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.

- In a classic replication scheme, if the transaction log threshold is exceeded, sets an unavailable subscriber to the `failed` state before the available transaction log space is exhausted. Set the transaction log threshold by specifying the `STORE` clause with a `FAILTHRESHOLD` value in the `CREATE REPLICATION` or `ALTER REPLICATION` statements. For an example, see Using a Return Service in a Classic Replication Scheme.

  If a master database sets the subscriber database to the `failed` state, it drops all of the data for the failed subscriber from its transaction 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.

  However, 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.

The default threshold value is 0, which means "no limit." See Setting Connection Attributes for Logging for details about transaction log failure threshold values.

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 or master database is changed from `failed` to `stop`.

> **Note**
>
> For more information about database states, see Table 11-1.

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, as described in Subscriber Failures.

Suspending or Resuming Classic Replication in Response to Conflicts

With classic replication, you can specify the number of replication conflicts per second at the table level at which conflict reporting is suspended and the number of conflicts per second at which conflict reporting resumes with the `CONFLICT REPORTING SUSPEND` and `CONFLICT REPORTING RESUME` attributes.

See Resolving Replication Conflicts.

Configuring the Network

You should consider certain issues when replicating TimesTen data over a network.

- Network Bandwidth Requirements
- Replication in a WAN Environment
- Configuring Network Interfaces with the ROUTE Clause
- Configuring Network Interfaces When Not Using the ROUTE Clause
- Identifying the Local Host of a Replicated Database

Network Bandwidth Requirements

The network bandwidth required for TimesTen Classic 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 4-1 provides guidelines for calculating the size of replicated records.

Table 4-1 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

Table 4-1 (Cont.) Replicated Record Sizes

Record Type	Size
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 256 KB. See Copying Updates Between Databases.

Replication in a WAN Environment

TimesTen Classic 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.

See AIX Prerequisites or Linux Prerequisites in the *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide* for information about changing TCP/IP kernel parameters for better performance.

Configuring Network Interfaces with the ROUTE Clause

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.

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.

While 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 (other than the default) using the `ROUTE` clause.

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, unless you want replication to use the default interface. 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.

The syntax of the `ROUTE` clause is:

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

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.

- When using the ROUTE clause in an active standby pair, 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.
- When using the ROUTE clause in a classic replication scheme that defines dual masters, 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.

The following example shows how to configure multiple network interfaces for an active standby pair.

If host1 host is configured with a second interface accessible by the host1fast host name, 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 dsn1, 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;
```

The following example shows how to configure multiple network interfaces for a classic replication scheme.

If host1 host is configured with a second interface accessible by the host1fast host name, 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 want 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 tries to connect using subsequent MASTERIP addresses in order of priority immediately. However, if the connection to the subscriber fails for any other reason, replication tries to connect using each of the SUBSCRIBERIP addresses in order of priority before it tries the MASTERIP address with the next highest priority.

The following example shows how to configure network priority on an active standby pair.

If the host1 host is configured with two network interfaces at IP addresses 192.168.1.100 and 192.168.1.101, and the host2 host 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 dsn1, dsn2
 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;
```

The following example shows how to configure network priority for a classic replication scheme.

If the host1 host is configured with two network interfaces at IP addresses 192.168.1.100 and 192.168.1.101, and the host2 host 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;
```

Configuring Network Interfaces When Not Using the ROUTE Clause

You can configure replication so that it uses the correct host names and IP addresses for each host when not using the ROUTE clause.

- Identifying Database Hosts on UNIX or Linux Without Using the ROUTE Clause
- User-Specified Addresses for TimesTen Daemons and Subdaemons

Identifying Database Hosts on UNIX or Linux Without Using the ROUTE Clause

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 replication scheme configuration that does not use the ROUTE clause, you can configure the 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 Classic replication tries to connect to the IP addresses in the same order as returned by the `gethostbyname` call. It tries 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 Classic 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 Classic 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 or Linux 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` cannot 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 or Linux 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 uses 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 either the `CREATE ACTIVE STANDBY PAIR` or `CREATE REPLICATION` statements used to configure your replication scheme.

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 `timesten.conf` file to specify an address for communication among the agents and daemons by including a `listen_addr` option.

See Managing TimesTen Daemon Attributes in *Oracle TimesTen In-Memory Database Operations Guide*.

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 `listen_addr` option in the `timesten.conf` file, then any process can talk to the daemons and agents.
- If you set `listen_addr` 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 `listen_addr` 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 Classic 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*.

5

Administering an Active Standby Pair Without Cache Groups

You can set up, administer, and recover an active standby pair.

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

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

Setting Up an Active Standby Pair with No Cache Groups

There are certain procedures you can run to set up an active standby pair.

See Configuring an Active Standby Pair with One Subscriber for an example.

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

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

- Configuring the Network
- Connection Attributes for Replicated Databases
- Managing the Transaction Log on a Replicated Database

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 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.
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. See Duplicating a Database.
7. Start the replication agent on the standby database. See Starting and Stopping the Replication Agents.
8. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` built-in 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.
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.

Recovering from a Failure of the Active Database

There are certain procedures you can run to recover from a failure of an active database.

This section includes the following topics:

- Recovering When the Standby Database is Ready
- Failing Back to the Original Nodes

Recovering When the Standby Database is Ready

There are procedures you can perform to recover the active database when the standby database is available and synchronized with the active database.

- When Replication is Return Receipt or Asynchronous
- When Replication is Return Twosafe

When Replication is Return Receipt or Asynchronous

There are certain tasks you can run to recover after an active database fails when replication uses return receipt or asynchronous replication.

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, call `ttRepStateSet('ACTIVE')`. This changes the role of the database from `STANDBY` to `ACTIVE`.
3. On the new active database, call `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.

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

There are certain procedures to run when recovering an active database when replication uses return twosafe.

Complete the following tasks:

1. On the standby database, call `ttRepStateSet('ACTIVE')`. This changes the role of the database from `STANDBY` to `ACTIVE`.
2. On the new active database, call `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.
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.

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 `ttRepStateGet` built-in procedure.

Failing Back to the Original Nodes

After a successful failover, you may want 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.

Recovering from a Failure of the Standby Database

There are certain tasks you can perform to recover a 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` built-in procedure with a *`localAction`* parameter of `2` (`COMMIT`) and commit the transaction again. For example:

   ```
   call ttRepSyncSet( null, null, 2);
   commit;
   ```

3. Call `ttRepStateSave('FAILED','`*`standby_database`*`','`*`host_name`*`')` on the active database. Afterwards, 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.
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.

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` built-in procedure.

Recovering After a Dual Failure of Both Active and Standby Databases

If both the active and standby databases fail, you can choose how to recover both databases.

Consider the following scenarios:

- The standby database fails. Then, 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.) 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 an Active Database

Perform a few tasks to recover an 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.
3. Call `ttRepStateSet('ACTIVE')` on the newly recovered database.
4. Continue with Step 6 in Setting Up an Active Standby Pair with No Cache Groups.

Recover a Standby Database

Perform a few tasks to recover a 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.
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. Call `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.
7. Continue from Step 6 in Setting Up an Active Standby Pair with No Cache Groups.

Recovering from the Failure of a Subscriber Database

You can recover a subscriber database if it fails.

To recover, either:

- 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.

Reversing the Roles of the Active and Standby Databases

You can 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. Call `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.
4. Call `ttRepDeactivate` on the current active database. This puts the database in the `IDLE` state.
5. Call `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` built-in 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.

6 Administering an Active Standby Pair with Cache Groups

You can replicate tables within either a read-only cache group or an asynchronous writethrough (AWT) cache group as long as they are configured within an active standby pair.

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

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

- Replicating Cache Groups Within Active Standby Pairs
- Changing User Names or Passwords Used by Replication
- Recovering from a Failure of the Active Database
- Recovering from a Failure of the Standby Database
- Recovering After a Dual Failure of Both Active and Standby Databases
- Recovering from the Failure of a Subscriber Database
- Reversing the Roles of the Active and Standby Databases
- Detecting Dual Active Databases
- Using a Disaster Recovery Subscriber in an Active Standby Pair

Replicating Cache Groups Within Active Standby Pairs

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 and Replicating a Read-Only Cache Group to understand the benefits of using cache groups within an active standby pair.

> **Note**
>
> TimesTen does not support replication of a user managed cache group or a synchronous writethrough (SWT) cache group in an active standby pair.

The following sections describe how to set up an active standby pair with cache groups:

- Setting Up an Active Standby Pair with an AWT Cache Group
- Setting Up an Active Standby Pair with a Read-Only Cache Group

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.

Setting Up an Active Standby Pair with an AWT Cache Group

With the active standby pair replication scheme, you must replicate all AWT cache groups. You can create the cache groups on the active master either before or after the active standby pair replication scheme is created.

- Before the creation of active standby pair: You can create cache groups on the active master before the active standby pair replication scheme is created. Then, create the active standby pair replication scheme and perform a distribution to include the cache groups. For detailed instructions on creating AWT cache group and then including it within a new active standby pair , see Replicating Cache Tables in TimesTen in *Oracle TimesTen In-Memory Database Cache Guide*.
- After creation of active standby pair: You can create cache groups on the active master after the active standby pair replication scheme is created. Then, perform a distribution to include the cache groups. See Making Other Changes to an Active Standby Pair.

Either way, cache groups are added to the active standby pair replication scheme after you perform a distribution to include these cache groups.

Setting Up an Active Standby Pair with a Read-Only Cache Group

With the active standby pair replication scheme, you can choose to replicate any read-only cache groups. You can create the cache groups on the active master either before or after the active standby pair replication scheme is created.

- Before the creation of active standby pair: You can create cache groups on the active master before the active standby pair replication scheme is created. Then, create the active standby pair replication scheme and perform a distribution to include the cache groups. See Creating a Read-Only Cache Group to Include Within a New Active Standby Pair.
- After creation of active standby pair: You can create cache groups on the active master after the active standby pair replication scheme is created. See Making Other Changes to an Active Standby Pair.

Either way, cache groups are added to the active standby pair replication scheme after you perform a distribution to include these cache groups.

Creating a Read-Only Cache Group to Include Within a New Active Standby Pair

You can set up an active standby pair that replicates cache tables in a read-only cache group.

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

- Configuring the Network
- Connection Attributes for Replicated Databases
- Managing the Transaction Log on a Replicated Database

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.
2. Create a database.
3. Set the cache administration user ID and password by calling the `ttCacheUidPwdSet` built-in procedure. For example:

   ```
   call ttCacheUidPwdSet('cacheadmin','orapwd');
   ```

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

   ```
   call ttCacheStart;
   ```

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

   ```
   CREATE READONLY CACHE GROUP customer_orders
           AUTOREFRESH INTERVAL 5 SECONDS
           FROM sales.customer
           (cust_num NUMBER(6) NOT NULL,
            region VARCHAR2(10),
            name VARCHAR2(50),
            address VARCHAR2(100),
            PRIMARY KEY(cust_num)),
           sales.orders
           (ord_num NUMBER(10) NOT NULL,
            cust_num NUMBER(6) NOT NULL,
            when_placed DATE NOT NULL,
            when_shipped DATE NOT NULL,
            PRIMARY KEY(ord_num),
            FOREIGN KEY(cust_num) REFERENCES sales.customer(cust_num));
   ```

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 running the `ttIsql cachegroups` 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:

   ```
   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:

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

9. Set up the replication agent policy for `master1` and start the replication agent. See Starting and Stopping the Replication Agents.
10. Load the cache group by using the `LOAD CACHE GROUP` statement. This starts the autorefresh process. For example:

    ```
    LOAD CACHE GROUP customer_orders 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. `ttRepAdmin` prompts for the values of `-uid`, `-pwd`, `-cacheuid` and `-cachepwd`.

    ```
    ttRepAdmin -duplicate -from master1 -host node1 -keepCG
     -connStr "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.
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:

    ```
    call ttRepStateGet;
    ```

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

    ```
    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. For example:

    ```
    ttRepAdmin -duplicate -from master2 -host node2 -nokeepCG
     -connStr "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.

Changing User Names or Passwords Used by Replication

In the active standby pair, you can modify either the TimesTen user name or password or (if there are cache groups in the active standby pair) the user names and passwords for the TimesTen cache administration user or its companion Oracle cache administration user.

When the `DDLReplicationLevel` connection attribute is 2 or larger, changes to the user names or passwords executed on the active master are automatically replicated to the standby master and any subscribers. When the `DDLReplicationLevel` connection attribute is 1, changes to the user names or passwords executed on the active master are not automatically replicated to the standby master and any subscribers. In this case, you must manually run each SQL statement on the active master, standby master, and any subscribers.

> **Note**
>
> See Making DDL Changes in an Active Standby Pair.

Perform the following to change any of the user names or passwords for the TimesTen user or, if there are cache groups in the active standby pair, for the TimesTen cache administration user or the Oracle cache administration user:

1. If you want to modify a password of a TimesTen user, use the `ALTER USER` statement on the active master database. If you want to change the TimesTen user name, you must first

drop all objects that the TimesTen user owns before dropping the user name and creating a new user.

To modify the password of the `sales` schema owner:

> **Note**
>
> See Creating or Identifying a Database User in *Oracle TimesTen In-Memory Database Operations Guide*.

```
ALTER USER sales IDENTIFIED BY newpwd;
```

2. If you want to modify any of the user names or passwords used for cache operations (such as the cache administration user), perform the instructions provided in Changing Cache User Names and Passwords in the *Oracle TimesTen In-Memory Database Cache Guide*.

Recovering from a Failure of the Active Database

If the active master has failed and the standby database did not fail or has recovered after a failure, then you can recover the active standby pair by making the standby master the new active master.

In addition, you can then swap the active and standby masters again so that they exist on the original nodes.

- Recovering When the Standby Database Is Ready
- Failing Back to the Original Nodes

> **Note**
>
> If both the active and standby masters fail, see Recovering After a Dual Failure of Both Active and Standby Databases.

Recovering When the Standby Database Is Ready

The first two sections describe how to recover the active database when the standby database is available and synchronized with the active database. The last section describes what to do if following the instructions from either of the first two sections fails; the standby database is available, but the data is not fully synchronized.

- When Replication Is Return Receipt or Asynchronous
- When Replication Is Return Twosafe
- When There Is Unsynchronized Data in the Cache Groups

When Replication Is Return Receipt or Asynchronous

You can failover to a standby database when the active fails.

Complete the following tasks:

1. On the standby database, stop the replication agent if it has not already been stopped.

2. On the standby database, call `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, call `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. On the new active database, start the replication agent and the cache agent.
5. Destroy the failed database (the old active) with the `ttDestroy` utility.
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` options with `ttRepAdmin` to recover and to preserve the cache group after the active master failure. See Duplicating a Database.
7. Set up the replication agent policy on the new standby database and start the replication agent. See Starting and Stopping the Replication Agents.
8. Start the cache agent on the new standby database.

Note

If any of these steps failed, follow the directions in When There Is Unsynchronized Data in the Cache Groups.

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

You can failover to a standby database when the active fails.

Complete the following tasks:

1. Stop the replication agent on the standby database if it has not already been stopped.
2. On the standby database, call `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, call `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. On the new active database, start the replication agent and the cache agent.
5. 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. If you are replicating a read-only cache group, the autorefresh state is automatically set to `PAUSED`.
6. 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.
7. Verify that the cache agent for the failed database has restarted. If it has not restarted, then start the cache agent.

> **Note**
>
> If any of these steps failed, follow the directions in When There Is Unsynchronized Data in the Cache Groups.

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 `ttRepStateGet` built-in procedure.

When There Is Unsynchronized Data in the Cache Groups

You can failover to a standby database when the active fails, even if there is unsynchronized data in the cache groups.

If the steps in either When Replication Is Return Receipt or Asynchronous or When Replication Is Return Twosafe fail, then there could be unsynchronized data in the AWT cache groups that has not been propagated to the Oracle database. In addition, there could be unsynchronized data on the Oracle database that has not been uploaded to any read-only cache groups that are included in the active standby pair replication scheme.

If there is data in any AWT cache groups on the standby master that has not been propagated when the active database failed, then simply recovering the standby database as the new active database is not an option. In this case, perform the following:

1. On the standby database, stop the replication agent and drop the replication configuration using the `DROP ACTIVE STANDBY PAIR` statement.

2. Stop the cache agent to ensure that no more updates are applied to the AWT cache groups while performing this recovery operation and to ensure that you control when any read-only cache groups that were included in the replication scheme are refreshed.
3. For any read-only cache groups that are included in the replication scheme, set the autorefresh state to pause with the `ALTER CACHE GROUP` ... `SET AUTOREFRESH STATE PAUSED` statement.
4. On the standby database, flush any unpropagated committed inserts or updates on TimesTen cache tables for any AWT cache groups to the cached Oracle Database tables, as follows:
   a. Set autocommit to off.
   b. Call the `ttCacheAllowFlushAwtSet` built-in procedure with the parameter set to 1. This built-in procedure enables you to run a `FLUSH CACHE GROUP` statement against an AWT cache group and should only be used in this recovery scenario.

      ```
      call ttCacheAllowFlushAwtSet(1);
      ```
   c. Run the `FLUSH CACHE GROUP` SQL statement against each AWT cache group to ensure that all data is propagated to the Oracle database.

      Note

      Running the `FLUSH CACHE GROUP` statement under these conditions on the AWT cache group only flushes the contents of the tables in the AWT cache group; that is, the data that was either inserted or updated. It does not take into account any delete operations. So, you may have rows that exist on the Oracle database that were deleted from the AWT cache group. It is up to the user to recover any delete operations.
   d. Call the `ttCacheAllowFlushAwtSet` built-in procedure with the parameter set to 0 to disallow any future running of the `FLUSH CACHE GROUP` statement on an AWT cache group.

      ```
      call ttCacheAllowFlushAwtSet(0);
      ```
   e. Commit after calling the `ttCacheAllowFlushAwtSet` built-in procedure with the parameter set to 0. You can also choose to reset autocommit to on, as it only needed to be off for the `ttCacheAllowFlushAwtSet` built-in procedure.
5. Drop and re-create all AWT cache groups using the `DROP CACHE GROUP` and `CREATE CACHE GROUP` statements.
6. Start the replication agent and the cache agent, since the cache agent needs to be active to refresh any read-only cache groups and both must be active in order to load the AWT cache groups.
7. Refresh all read-only cache groups using the `REFRESH CACHE GROUP` statement to upload most current committed data from the cached Oracle database tables. Use the `REFRESH CACHE GROUP ... PARALLEL` *n* clause to concurrently load these cache groups over multiple threads.
8. Load all AWT cache groups using the `LOAD CACHE GROUP` statement to begin the autorefresh process. Use the `LOAD CACHE GROUP ... PARALLEL` *n* clause to concurrently load these cache groups over multiple threads.
9. Stop both the replication agent and the cache agent in preparation to re-create the active standby pair.

10. Re-create the replication configuration on the standby database using the `CREATE ACTIVE STANDBY PAIR` statement.
11. Set the old standby database as the new active database, destroy the failed old active database, perform a duplicate of the active to create a new standby database, and start the cache and replication agents on the standby as described in the steps listed in When Replication Is Return Receipt or Asynchronous.

Failing Back to the Original Nodes

After a successful failover, you may want 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.

Recovering from a Failure of the Standby Database

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

1. 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` built-in procedure with a *`localAction`* parameter of `2` (`COMMIT`) and commit the transaction again. For example:

   ```
   Command> call ttRepSyncSet( null, null, 2);
   Command> commit;
   ```

2. Call `ttRepStateSave('FAILED','`*`standby_database`*`','`*`host_name`*`')` on the active database. Then, as long as the standby database is unavailable, updates to the active database are replicated directly to the subscriber databases. Additional subscriber databases may also be duplicated directly from the active.
3. Recover the standby database in one of the following ways:
   a. Connect to the standby database. This triggers recovery from the local transaction logs. If the standby database recovers, go to Step 4; otherwise, continue to Step 3b.
   b. Destroy the current version of the standby database with the `ttDestroy` utility.
   c. Duplicate a new 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` options with `ttRepAdmin` to recover and to preserve the cache group after the standby master failure. See Duplicating a Database.
4. Set up the replication agent policy and start the replication agent on the standby database. See Starting and Stopping the Replication Agents.
5. Start the cache agent on the standby database.

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` built-in procedure.

Recovering After a Dual Failure of Both Active and Standby Databases

If both the active and standby databases fail at around the same time and if you can reconnect to both of them almost immediately, then restart the replication agents (and cache agents if applicable) and continue.

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.
3. Call `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. Connect to the failed standby master 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`.
6. Verify that the replication agent for the failed standby database has restarted. If it has not restarted, then start the replication agent. See Starting and Stopping the Replication Agents.
7. Verify that the cache agent for the failed standby database has restarted. If it has not restarted, then start the cache agent.

Alternatively, consider the following scenarios where both the active and standby master databases fail:

- 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.) The new active database fails before the new standby database is fully synchronized with it.

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

In this case, you could potentially perform one of the following options:

- Recover the Active Database and Duplicate a New Standby Database
- Recover the Standby Database to Be the New Active Master
- Restore the Active Master from a Backup

Recover the Active Database and Duplicate a New Standby Database

You can recover an active database and then duplicate it to a new standby 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`.

   > **Note**
   >
   > If this fails, perform the steps listed in Restore the Active Master from a Backup..

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.
3. Call `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. See Duplicating a Database.
6. Set up the replication agent policy on the standby database and start the replication agent. See Starting and Stopping the Replication Agents.
7. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` built-in procedure to check the state.
8. Start the cache agent for on the standby database using the `ttCacheStart` built-in procedure or the `ttAdmin -cacheStart` utility.
9. Duplicate all of the subscribers from the standby database. See Duplicating a Master Database to a Subscriber. 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.

Recover the Standby Database to Be the New Active Master

1. Connect to the failed standby master 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`.

   > **Note**
   >
   > If this fails, perform the steps listed in Restore the Active Master from a Backup.

2. If the replication agent for the failed standby master has automatically restarted, stop the replication agent. See Starting and Stopping the Replication Agents.
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. Call `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.
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. See Duplicating a Database.
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.
12. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` built-in procedure to check the state.
13. Start the cache agent for the standby database using the `ttCacheStart` built-in procedure or the `ttAdmin -cacheStart` utility.
14. Duplicate all of the subscribers from the standby database. See Duplicating a Master Database to a Subscriber. 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.

Restore the Active Master from a Backup

If both the active and standby masters fail and neither comes up, you can restore the active master if you have a backup.

1. Restore the active master from a backup, as described in Backing Up and Restoring a TimesTen Database with Cache Groups in the *Oracle TimesTen In-Memory Database Cache Guide*.
2. Drop the replication configuration using the `DROP ACTIVE STANDBY PAIR` statement.
3. Drop and re-create all AWT cache groups using the `DROP CACHE GROUP` and `CREATE CACHE GROUP` statements.
4. Start the replication agent and the cache agent, since the cache agent needs to be active to refresh any read-only cache groups and both must be active in order to load the AWT cache groups.
5. Refresh all read-only cache groups using the `REFRESH CACHE GROUP` statement to upload most current committed data from the cached Oracle database tables. Use the `REFRESH CACHE GROUP ... PARALLEL` *n* clause to concurrently load these cache groups over multiple threads.

6. Load all AWT cache groups using the `LOAD CACHE GROUP` statement to begin the autorefresh process. Use the `LOAD CACHE GROUP ... PARALLEL` *n* clause to concurrently load these cache groups over multiple threads.
7. Stop both the replication agent and the cache agent in preparation to re-create the active standby pair.
8. Re-create the replication configuration using the `CREATE ACTIVE STANDBY PAIR` statement.
9. Call `ttRepStateSet('ACTIVE')` on the active 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.
10. Set up the replication agent policy and start the replication agent on the active database. See Starting and Stopping the Replication Agents.
11. Start the cache agent on the active database.
12. 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.
13. 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.
14. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` built-in procedure to check the state.
15. Start the cache agent for the standby database using the `ttCacheStart` built-in procedure or the `ttAdmin -cacheStart` utility.
16. Duplicate all of the subscribers from the standby database. See Duplicating a Master Database to a Subscriber. Use the `-noKeepCG` command line option with `ttRepAdmin` in order to convert the cache group to regular TimesTen tables on the subscribers.
17. 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.

Recovering from the Failure of a Subscriber Database

There are a couple of methods you can use to recover from the failure of a subscriber database.

- 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.

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. Call `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.
4. On the active database, for all read-only cache groups with autorefresh state set to `ON`, set the autorefresh state to `PAUSED`.
5. Call `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. Call `ttRepStateSet('ACTIVE')` on the current standby database. This database now acts as the active database in the active standby pair.
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` built-in 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.

Detecting Dual Active Databases

There is no difference for active standby pairs that replicate cache groups.

See Detection of Dual Active Databases.

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
- Rolling Out a Disaster Recovery Subscriber
- Switching over to the Disaster Recovery Site

- Returning to the Original Configuration at the Primary Site

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 the Oracle database 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
    -connStr "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 want 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.

   See Setting the Transaction Log Failure Threshold.

4. Start the replication agent for the disaster recovery subscriber using the `ttRepStart` built-in 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.

   See Starting and Stopping the Replication Agents.

Switching over to the Disaster Recovery Site

When the primary site has failed, you can switch over to the disaster recovery site.

There are one of two ways to switch over to the disaster recovery site.

- Creating a New Active Standby Pair After Switching to the Disaster Recovery Site: 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.
- Switching over to a Single 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` built-in procedure or the `ttRepAdmin` command with the `-wait` option.

   ```
   Command> call 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` built-in 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` built-in procedure. For example, on the database `drsub` in this example, call:

   ```
   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.
12. Set up the replication agent policy on the standby database and start the replication agent. See Starting and Stopping the Replication Agents.

13. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` built-in procedure to check the state.
14. Start the cache agent for the standby database using the `ttCacheStart` built-in procedure or the `ttAdmin -cacheStart` utility.
15. Duplicate all of the subscribers from the standby database. See Duplicating a Master Database to a Subscriber. 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.

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` built-in 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` built-in procedure or the `ttAdmin` command with the `-repstart` option. For example, to start the replication agent for the database `drsub`, use:

   ```
   call ttRepStart;
   ```

   See Starting and Stopping the Replication Agents.
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, 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 want 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. 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. 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.

7

Altering an Active Standby Pair

There are certain procedures you can perform to alter an active standby pair.

- Making DDL Changes in an Active Standby Pair
- Making Other Changes to an Active Standby Pair

Making DDL Changes in an Active Standby Pair

You can configure automatic replication of certain DDL statements. When you run a supported DDL statement on the active master, it is automatically replicated to all databases in the active standby pair replication scheme.

- Controlling Replication of Objects to All Databases in an Active Standby Pair
- DDL Statements That Can Be Automatically Replicated
- Creating a New PL/SQL Object in an Existing Active Standby Pair
- Restrictions on Making DDL Changes in an Active Standby Pair
- Examples Showing How to Making DDL changes in an Active Standby Pair

Controlling Replication of Objects to All Databases in an Active Standby Pair

Use the `DDLReplicationLevel` and `DDLReplicationAction` connection attributes to control what objects that are created or dropped by DDL statements are replicated to the databases involved in an active standby pair replication scheme. For more information on connection attributes, see Specifying Data Source Names to Identify TimesTen Databases in the *Oracle TimesTen In-Memory Database Operations Guide*

The `DDLReplicationLevel` connection attribute controls what DDL statements are replicated.

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

  To include tables in the replication scheme, the `DDLReplicationAction` connection attribute must be set to `'INCLUDE'` (the default) before creating the table. If `DDLReplicationAction='EXCLUDE'`, then the table is not included in the replication scheme. If the table is excluded from the replication scheme, then the DDL statements for creating and dropping tables are replicated to the standby master; however, any DML statements run on the table are not replicated.

Note

You may want to exclude a table from a replication scheme if:

- You want to create a table in the replication scheme without either a primary key or a unique index on non-nullable columns.
- You want to create a temporary table where the data is only used locally and you do not want to replicate updates for this table.

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

However, you cannot alter a table to add a `NOT NULL` column to a table that is part of a replication scheme with the `ALTER TABLE ... ADD COLUMN NOT NULL DEFAULT` statement. You must remove the table from the replication scheme first before you can add a `NOT NULL` column to it.

- `DDLReplicationLevel=3` enables the following:
  - Replication to all databases in the replication scheme of the same objects that are replicated when `DDLReplicationLevel=2`.
  - Replication of creating and dropping of views to all databases in the replication scheme.
  - Replication of creating and dropping of sequences to all databases in the replication scheme, as long as `DDLReplicationAction='INCLUDE'` (the default) before creation. If `DDLReplicationAction='EXCLUDE'`, then the sequence is not included in the replication scheme. If the sequence is excluded from the replication scheme, then the DDL statements for creating and dropping sequences are replicated to the standby master; however, each sequence on the active master and standby master are separate objects.
  - Replication of the results to the standby master when you set the cache administration user name and password on the active master with the `ttCacheUidPwdSet` built-in procedure. You do not need to stop and restart the cache agent or replication agent in order to run the `ttCacheUidPwdSet` built-in procedure on the active master. See Changing User Names or Passwords Used by Replication.
  - With this level, you can alter a table to add a `NOT NULL` column to a table that is part of a replication scheme with the `ALTER TABLE ... ADD COLUMN NOT NULL DEFAULT` statement.

You can set the `DDLReplicationLevel` attribute as part of the connection string or after the connection starts with the `ALTER SESSION` statement:

```
ALTER SESSION SET DDL_REPLICATION_LEVEL=3;
```

You can set the `DDLReplicationAction` attribute as part of the connection string or after the connection starts with the `ALTER SESSION` statement:

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

See ALTER SESSION and ALTER ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference*.

Note

DDL operations are automatically committed. When `RETURN TWOSAFE` has been specified, errors and timeouts may occur as described in RETURN TWOSAFE. If a `RETURN TWOSAFE` timeout occurs, the DDL transaction is committed locally regardless of the LOCAL COMMIT ACTION that has been specified.

DDL Statements That Can Be Automatically Replicated

You can run the following DDL statements in an active standby pair without stopping the replication agent. In addition, these statements are replicated to all databases in the replication scheme.

The following statements are automatically replicated when `DDLReplicationLevel` is 2 or 3 (the default is 2):

- Create, alter, or drop a user with the `CREATE USER`, `ALTER USER`, or `DROP USER` statements.
- Grant or revoke privileges from a user with the `GRANT` or `REVOKE` statements.
- Alter a table to add or drop a column with the `ALTER TABLE ... ADD COLUMN` or `ALTER TABLE ... DROP COLUMN` statements.

  These are the only `ALTER TABLE` clauses that are replicated. However, when `DDLReplicationLevel=2`, you cannot alter a table to add a `NOT NULL` column to a table that is part of a replication scheme with the `ALTER TABLE ... ADD COLUMN NOT NULL DEFAULT` statement. You can run this statement if `DDLReplicationLevel=3`.
- Create or drop a table with the `CREATE TABLE` or `DROP TABLE` statements. The new table is included in the active standby pair.
- Create or drop a synonym with the `CREATE SYNONYM` or `DROP SYNONYM` statements.
- Create or drop an index with the `CREATE INDEX` or `DROP INDEX` statements.

You can perform the following tasks in an active standby pair without stopping the replication agent. In addition, these statements are replicated to all databases in the replication scheme. The following statements are automatically replicated when `DDLReplicationLevel` is set to 3:

- Create or drop a view with the `CREATE VIEW` or `DROP VIEW` statements.
- Create or drop a sequence with the `CREATE SEQUENCE` or `DROP SEQUENCE` statements. These statements are automatically replicated to all databases in the replication scheme and included in the active standby pair when the `DDLReplicationAction` connection attribute is set to `INCLUDE` (the default) before creating the sequence; the sequence is not included in the replication scheme if the `DDLReplicationAction` connection attribute is set to `EXCLUDE`.

You do not have to stop the cache agent or replication agent when you set the user name and password for the cache administration user on the active master with the `ttCacheUidPwdSet` built-in procedure. When `DDLReplicationLevel=3`, then this information is automatically replicated to the standby master. See Changing User Names or Passwords Used by Replication.

You can perform the following tasks in an active standby pair only after stopping the replication agents. These statements are not replicated to the standby master, so you must ensure that the changes are propagated to the standby master and any subscribers by either performing a duplicate or running these statements on all nodes in the replication scheme after the replication agents are stopped. After processing completes, restart the replication agents on all nodes.

- The DDL statements for creating, dropping, or altering a materialized view.
- Changing the autorefresh mode or interval using the `ALTER CACHE GROUP` ... `SET AUTOREFRESH MODE` or `ALTER CACHE GROUP` ... `SET AUTOREFRESH INTERVAL` statements.

You can perform the following tasks in an active standby pair without stopping the replication agent. However, these statements are not replicated to the standby master, so you must ensure that the changes are propagated to the standby master and any subscribers by either performing a duplicate or running these statements on all nodes in the replication scheme.

- Changing the autorefresh state of a cache group using the `ALTER CACHE GROUP` ... `SET AUTOREFRESH STATE` statement. However, you cannot set a cache group autorefresh state to `OFF` on the active master.
- Create or drop a PL/SQL function, PL/SQL procedure, PL/SQL package, or PL/SQL package body. You do not need to stop the replication agents for these objects. See Creating a New PL/SQL Object in an Existing Active Standby Pair.
- Any other DDL statements that are not replicated (except for materialized views).

Creating a New PL/SQL Object in an Existing Active Standby Pair

There are certain tasks to perform to add a new PL/SQL procedure, package, package body or function to an existing active standby pair.

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 and any subscribers.
3. Grant privileges to the new PL/SQL object on the active database. The `GRANT` statement is replicated to the standby database and any subscribers.

Restrictions on Making DDL Changes in an Active Standby Pair

There are certain restrictions when making DDL changes in an active standby pair.

When `DDLReplicationLevel`=2 or 3:

- `CREATE TABLE ... AS SELECT`, `ALTER TABLE ... ADD CONSTRAINT`, `ALTER TABLE ... ADD UNIQUE` and `ALTER TABLE ... MODIFY` statements are not replicated.
- The `CREATE INDEX` statement is replicated only when the index is created on an empty table. To create a new index on populated tables, set `DDLReplicationLevel` to a value less than 2 and create the index manually on both the active and standby.
- These statements cannot run on the standby database when `DDLReplicationLevel`=2 or 3:
  - `CREATE USER`, `ALTER USER`, `DROP USER`
  - `CREATE TABLE`, `DROP TABLE`
  - `CREATE INDEX`, `DROP INDEX`
  - `GRANT`, `REVOKE`
  - `CREATE SYNONYM`, `DROP SYNONYM`

When `DDLReplicationLevel`=3:

- `CREATE SEQUENCE ... CYCLE` statements are not replicated.
- These statements cannot run on the standby database when `DDLReplicationLevel=3`:
  - `CREATE INDEX`, `DROP INDEX`
  - `CREATE SEQUENCE`, `DROP SEQUENCE` when `DDLReplicationAction='INCLUDE'`.

> **Note**
>
> However, you can create or drop a sequence on the standby master when `DDLReplicationLevel=3` if `DDLReplicationAction='EXCLUDE'`.

Examples Showing How to Making DDL changes in an Active Standby Pair

These examples demonstrate how to make DDL changes in an active standby pair.

The following example demonstrates DDL changes when you 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.

The following example demonstrates DDL changes when you 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.

The following example demonstrates DDL changes when `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 range index on columns:
    B
  TAB2: unique range index on columns:
    A
  2 indexes found.

Indexes on table TERRY.NEWTAB:
  NEWTAB: unique range index on columns:
    A
  1 index found.

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

The following example demonstrates DDL changes when `CREATE SYNONYM` is replicated.

The `DDLReplicationLevel` is already set to `2`, since it is the default. 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 before making certain 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 cache group after the active standby pair has already been created.

  Note: See Replicating Cache Groups Within Active Standby Pairs for details on how to add a cache group that already exists before you create the active standby pair.
- 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.
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 Showing How to Alter an Active Standby Pair.
4. Start the replication agent on the active database. See Starting and Stopping the Replication Agents.
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.
8. Set up the replication agent policy on the standby database and start the replication agent. See Starting and Stopping the Replication Agents.

9. Wait for the standby database to enter the `STANDBY` state. Use the `ttRepStateGet` built-in 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` built-in procedure or the `ttAdmin -cacheStart` utility.
11. Duplicate all of the subscribers from the standby database. See Duplicating a Master Database to a Subscriber. 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.
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.

Examples Showing How to Alter an Active Standby Pair

These examples demonstrate adding, dropping and changing an active standby pair.

The following example demonstrates adding a subscriber to an active standby pair

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

The following example demonstrates dropping subscribers from an active standby pair.

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

The following example demonstrates 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;
```

The following example demonstrates adding a cache group to an active standby pair.

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

8

Using Oracle Clusterware to Manage Active Standby Pairs

Oracle Clusterware monitors and controls applications to provide high availability. Oracle Clusterware is a general purpose cluster manager that manages and monitors the availability of software components that participate in a cluster.

There are procedures on how to use Oracle Clusterware to manage availability for the databases in an active standby pair replication scheme in TimesTen.

Note

See the Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation.

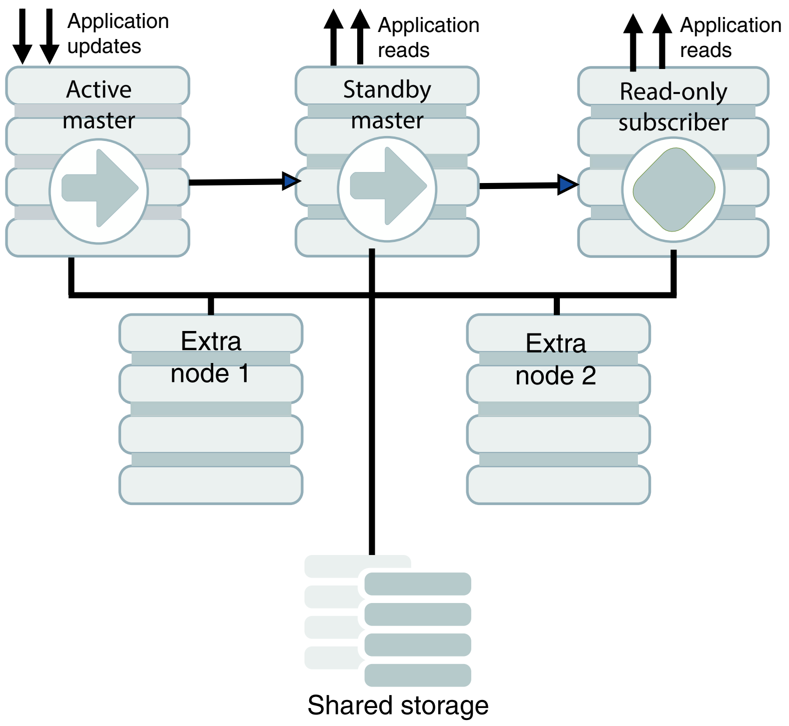
Overview of How Oracle Clusterware Can Manage TimesTen

Use Oracle Clusterware to manage only the following configurations for active standby pair replication schemes.

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

Figure 8-1 shows an active standby pair with one read-only subscriber in the same local network. The active master, the standby master 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 8-1 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 Clusterware Management and Recovering from Failures.

Oracle Clusterware can be implemented at two levels of availability for TimesTen.

- The *basic* level of availability manages two master nodes configured as an active standby pair 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 enables one of the extra nodes to take on the role of a failed node automatically.

Note

If your applications connect to TimesTen in a client/server configuration, automatic client failover enables the client to reconnect automatically to the active database after a failure. See Using Automatic Client Failover for an Active Standby Pair 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`. The information in this file is used to create Oracle Clusterware *resources*. Resources are used to manage the TimesTen daemon, TimesTen databases, TimesTen processes, user applications, and virtual IP addresses. You can run the `ttCWAdmin` utility from any host in the cluster, as long as the `cluster.oracle.ini` file is reachable and readable from this host. 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 Configuring Oracle Clusterware Management with the cluster.oracle.ini File.

Requirements, Considerations, and Installation for Your Cluster

There are requirements and installation steps when creating your cluster.

- Required Privileges
- Hardware and Software Requirements
- Install Oracle Clusterware
- Install TimesTen on Each Host
- Register the TimesTen Cluster Information

Required Privileges

There are privileges required to run `ttCWAdmin` commands.

See ttCWAdmin in *Oracle TimesTen In-Memory Database Reference*.

Hardware and Software Requirements

TimesTen supports Clusterware on Linux or UNIX platforms except Linux arm64.

TimesTen supports Oracle Clusterware with TimesTen active standby pair replication. See Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation 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 or Linux primary group as the Oracle Clusterware installation owner.

> **Note**
>
> 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. When adjusting the system clocks on any nodes to be synchronized with each other, do not set any clock backward in time.

Install Oracle Clusterware

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

For example, see the Oracle Grid Infrastructure Grid Infrastructure Installation and Upgrade Guide.

Oracle Clusterware starts automatically after successful installation.

Note

You can verify whether Oracle Clusterware is running on all hosts in the cluster by running the following:

```
crsctl check crs -all
```

Install TimesTen on Each Host

Use the `ttInstanceCreate` command to install TimesTen in the same location on each host in the cluster, including extra hosts.

See Create an Instance Interactively for Oracle Clusterware in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

When responding to the various prompts, note that:

- 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 or Linux primary group as the Oracle Clusterware installation owner.

In addition, when you respond yes to the following question:

```
Would you like to use TimesTen Replication with Oracle Clusterware?
```

Then, the `ttInstanceCreate` command prompts you for values used for Oracle Clusterware, each of which is stored in the `ttcrsagent.options` file:

- The TCP/IP port number associated with the TimesTen cluster agent (`ttCRSAgent`). The port number must be the same on all nodes of the cluster. If you do not provide a port number, then TimesTen adds six to the default TimesTen daemon port number to be the TCP/IP port number associated with the TimesTen cluster agent. Thus, the default daemon port number associated with the TimesTen cluster agent is 3574 for 64-bit systems.
- 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.

See Installing Oracle Clusterware for Use with TimesTen and Create an Instance Interactively for Oracle Clusterware in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

The `ttCWAdmin –init` and `ttCWAdmin –shutdown` commands use the `ttcrsagent.options` file to initiate and shut down the TimesTen cluster. The `ttcrsagent.options` file is located in the TimesTen daemon home directory.

You should not manually alter the `ttcrsagent.options` file. Instead, use the `ttInstanceModify -crs` command to create or modify the information in this file after the TimesTen cluster has been initiated. You can also use the `-record` and `-batch` options for `setup.sh` to perform identical installations on additional hosts.

Note

See Change the Oracle Clusterware Configuration for an Instance in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

The current home location of Oracle Clusterware is set in the `CRS_HOME` environment variable. In addition, the `ttInstanceModify -crs` command shows the current location of the Oracle Clusterware home as part of the prompts.

Note

See Start the TimesTen Cluster Agent for more information on the `ttcrsagent.options` file. For more information about `ttInstanceCreate` and `ttInstanceModify`, see ttInstanceCreate and ttInstanceModify respectively in *Oracle TimesTen In-Memory Database Reference*.

The following example shows how the `ttInstanceModify -crs` prompts for you to modify each item in the `ttcrsagent.options` file:

```
% ttInstanceModify -crs

Cannot find instance_info file : /etc/TimesTen/instance_info

Would you like to modify the existing TimesTen Replication with Oracle
Clusterware configuration? [ no ] yes

This TimesTen instance is configured to use an Oracle Clusterware installation
located in : /mydir/oracle/crs/app/11.2.0
Would you like to change this value? [ no ] no

The TimesTen Clusterware agent is configured to use port 54504
Would you like to change this value? [ no ] no

The TimesTen Clusterware agent is currently configured with these nodes :

node1
node2
node3
node4

Would you like to change these values? [ no ]

Overwrite the existing TimesTen Clusterware options file? [ no ] no
```

Register the TimesTen Cluster Information

TimesTen cluster information is stored in the Oracle Cluster Registry (OCR).

As the root user, enter this command:

```
ttCWAdmin -ocrConfig
```

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

Restricted Commands and SQL Statements

When you use Oracle Clusterware with TimesTen, the active standby pair replication scheme is created on the active database with the `ttCWAdmin -create` command and dropped with the `ttCWAdmin -drop` command.

In between the `ttCWAdmin -create` and `ttCWAdmin -drop` commands, you cannot run certain commands or SQL statements.

However, you can perform these commands or SQL statements when you use the `ttCWAdmin -beginAlterSchema` and the `ttCWAdmin -endAlterSchema` commands, as described in Changing the Schema.

You cannot run the following commands or SQL statements:

- Creating, altering, or dropping the active standby pair with the `CREATE ACTIVE STANDBY PAIR`, `ALTER ACTIVE STANDBY PAIR`, and `DROP ACTIVE STANDBY PAIR` SQL statements.
- Starting or stopping the replication agent with either the `-repStart` and `-repStop` options of the `ttAdmin` utility or the `ttRepStart` or `ttRepStop` built-in procedures. For more information, see Starting and Stopping the Replication Agents.
- Starting or stopping the cache agent after the active standby pair has been created with either the `-cacheStart` and `-cacheStop` options of the `ttAdmin` utility or the `ttCacheStart` and `ttCacheStop` built-in procedures.
- Duplicating the database with the `-duplicate` option of the `ttRepAdmin` utility.

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 specified in the `cluster.oracle.ini` file.

Creating and Initializing a Cluster

There are procedures to create and initialize a cluster.

- Start the TimesTen Cluster Agent
- Create and Populate a TimesTen Database on One Host
- Create System DSN Files on Other Hosts
- Create a cluster.oracle.ini File
- Create the Oracle Clusterware Resources to Manage Virtual IP Addresses
- Create an Active Standby Pair Replication Scheme
- Start the Active Standby Pair and the Applications
- Load Cache Groups
- Include More Than One Active Standby Pair in a Cluster
- Configure an Oracle Database as a Disaster Recovery Subscriber
- Configure a Read-Only Subscriber That Is Not Managed by Oracle Clusterware

If you plan to have more than one active standby pair in the cluster, see Include More Than One Active Standby Pair in a Cluster.

If you want to configure an Oracle database as a remote disaster recovery subscriber, see Configure an Oracle Database as a Disaster Recovery Subscriber.

If you want to set up a read-only subscriber that is not managed by Oracle Clusterware, see Configure a Read-Only Subscriber That Is Not Managed by Oracle Clusterware.

Start the TimesTen Cluster Agent

Start a TimesTen cluster agent (`ttCRSAgent`) and TimesTen cluster daemon monitor (`ttCRSDaemon`) on all hosts in the cluster by running the `ttCWAdmin -init` command.

You can run this command on any host in the cluster that is defined in the `ttcrsagent.options` file.

For example:

```
ttCWAdmin -init
```

The `ttCWAdmin -init` command performs the following:

- Reads the `ttcrsagent.options` file and launches the TimesTen main daemon on each of the hosts defined in this file.
- Starts and registers the TimesTen cluster agent (`ttCRSAgent`) and the TimesTen cluster daemon monitor (`ttCRSDaemon`) on the all hosts in the cluster. There is one TimesTen cluster agent and one TimesTen cluster daemon monitor for the TimesTen installation on each host. When the TimesTen cluster agent has started, Oracle Clusterware begins monitoring the TimesTen daemon on each host and restarts a TimesTen daemon if it fails.

To start and register the TimesTen cluster agent and the TimesTen cluster daemon monitor on specific hosts in the cluster, use the `-hosts` command to specify the desired hosts in the cluster to start.

```
ttCWAdmin -init -hosts "host1, host2"
```

Note

You must stop the TimesTen cluster agent on the local host with the `ttCWAdmin -shutdown` before you run a `ttDaemonAdmin -stop` command; otherwise the cluster agent restarts the TimesTen 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) and populate with data as appropriate. However, before you create cache groups, you must first decide when to load the cache groups.

- For best performance, load the cache group tables from the Oracle database tables before the `ttCWAdmin -create` command. There is less performance overhead when cache groups are loaded with initial data before the duplicate is performed on the active database to create the standby database (and any subscriber databases).

  For this option, perform the following:

1. Start the cache agent as follows:

   ```
   call ttCacheStart;
   ```

   > **Note**
   >
   > Since this is before the `ttCWAdmin -start` command, you can start the cache agent at this time. The `ttCWAdmin -start` command notes that the cache agent is already started and continues.

2. Use the `LOAD CACHE GROUP` statement to load the cache group tables from the Oracle database tables.
3. If using cache groups with autorefresh, set the autorefresh state to pause with the `ALTER CACHE GROUP SET AUTOREFRESH STATE PAUSED` statement. The autorefresh state will be set to `ON` as part of the `ttCWAdmin -start` command.

The following example demonstrates how to create a read-only cache group with autorefresh, load the data, and then set the autorefresh state to pause:

```
Command> call ttCacheStart;
Command> CREATE READONLY CACHE GROUP my_cg
  AUTOREFRESH MODE INCREMENTAL INTERVAL 60 SECONDS
  FROM t1 (c1 NUMBER(22) NOT NULL PRIMARY KEY, c2 DATE, c3 VARCHAR(30));

Command> LOAD CACHE GROUP my_cg COMMIT EVERY 100 ROWS PARALLEL 4;
Command> ALTER CACHE GROUP my_cg SET AUTOREFRESH STATE PAUSED;
```

- Alternatively, wait to load the cache group tables after the `ttCWAdmin -start` as described in Load Cache Groups. The data will be replicated to the standby database and any subscriber databases.

Create System DSN Files on Other Hosts

On all hosts that are to be included in the cluster, create the system DSN (`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 Configuring Oracle Clusterware Management with the cluster.oracle.ini File.

Create a cluster.oracle.ini File

Create a `cluster.oracle.ini` file as a text file.

See Configuring Oracle Clusterware Management with the cluster.oracle.ini File for details about its contents and acceptable locations for the file.

Create the Oracle Clusterware Resources to Manage Virtual IP Addresses

Advanced availability involves configuring spare master or subscriber hosts that are idle until needed to replace master or subscriber hosts (used in the active standby pair replication scheme) that either shut down unexpectedly or experience an unrecoverable error.

This is an optional step that is only necessary if you decide to configure advanced availability.

If you are planning on providing additional master or subscriber hosts for advanced availability, then you need to configure virtual IP addresses (one for each master host and subscriber actively used in the active standby pair). See Configuring Advanced Availability for more details on how many virtual IP addresses should be created.

In this case, perform the following:

1. Designate (or create) new virtual IP addresses on the network that are to be used solely for managing multiple hosts in a TimesTen replication environment managed by Oracle Clusterware.
2. Configure these VIP addresses for use to manage multiple hosts for advanced availability in the `cluster.oracle.ini` file, as described in Configuring Advanced Availability.
3. Create the Oracle Clusterware resources that manage these VIP addresses by running the `ttCWAdmin -createVIPs` command as the `root` user on any host in the cluster.

   For example:

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

   The VIP address names created by this command start with `network_` followed by the TimesTen instance name, TimesTen instance administrator, and the DSN. Whereas, the VIP addresses created for use by Oracle Clusterware are prefixed with `ora`.

Note

You must run the `ttCWAdmin -createVIPs` command before the `ttCWAdmin -create` command. If you decide that you want to use VIP addresses for advanced availability after you run the `ttCWAdmin -create` command, you must perform the following:

a. Run `ttCWadmin -drop` to drop the active standby pair replication scheme.
b. Add VIP addresses into `cluster.oracle.ini` file.
c. Run `ttCWadmin -createVIPs` to create the resources to manage the VIP addresses.
d. Run `ttCWAdmin -create` to create the active standby pair replication scheme managed by Oracle Clusterware.

Once created, the only way to drop the Oracle Clusterware resources that manage the VIP addresses is to run the `ttCWAdmin -dropVIPs` command. Before you can drop the virtual IP addresses, you must first run the `ttCWAdmin -drop` command.

The following is an example of how to drop the virtual IP addresses:

```
ttCWAdmin -dropVIPs -dsn myDSN
```

For an example of when to use the `ttCWAdmin -dropVIPs` command, see Removing an Active Standby Pair from a Cluster.

Create an Active Standby Pair Replication Scheme

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

> **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 Configuring Oracle Clusterware Management with the cluster.oracle.ini File.

For example:

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

The `ttCWAdmin -create` command prompts for the following:

- Prompts for the name of a TimesTen user with `ADMIN` privileges. If cache groups are being managed by Oracle Clusterware, enter the TimesTen cache administration user name.
- Prompts for the TimesTen password for the previously entered user name.
- If cache groups are being used, prompts for the password for the Oracle cache administration user. This password is provided in the `OraclePWD` connection attribute when the cache administration user connects.
- Prompts for a random string used to encrypt the above information.

If you want to specify the path and name of a file to be used as the `cluster.oracle.ini` file, use the `-ttclusterini` option of the `ttCWAdmin -create` command.

```
ttCWAdmin -create -dsn myDSN -ttclusterini path/to/cluster/mycluster.ini
```

To drop the active standby pair, use the `ttCWAdmin -drop` command, as follows:

```
ttCWAdmin -drop -dsn myDSN
```

> **Note**
>
> If your application connects to the TimesTen database using the virtual IP address, then this connection drops with the `ttCWAdmin -drop` command, since the virtual IP address is managed by Oracle Clusterware. However, if your application connects to the TimesTen database using the host name, the connection is not dropped.

For examples showing the sequence in which to use the `ttCWAdmin -create` and `ttCWAdmin -drop` commands, see Managing Active Standby Pairs in a Cluster and Managing Read-Only Subscribers in the Active Standby Pair.

Start the Active Standby Pair and the Applications

Start the cluster with the active standby pair replication scheme by running the `ttCWAdmin -start` command on any host.

This starts the cache agent (if not already started) and replication agent on the active database, performs the duplicate to create the standby database (and any subscriber databases), and starts the cache agent and replication agent on the standby (and any subscribers).

If you do not specify `-noApp` option, the applications are also started. If you do specify `-noApp` option, then you can start and stop the applications with the `-startApps` and `-stopApps` options respectively.

For example:

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

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

- TimesTen daemon monitor`ttCRSMaster`
- Active service `ttCRSActiveService`
- Standby service `ttCRSsubservice`
- Monitor for application AppName

The following example starts the cache and replication agents, but does not start the applications because of the inclusion of the `-noapp` option:

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

To start and stop applications, use the `ttCWAdmin -startApps` and `-stopApps` commands as shown below:

```
ttCWAdmin -startapps -dsn myDSN
```

```
ttCWAdmin -stopapps -dsn myDSN
```

To stop the TimesTen database monitor (`ttCRSMaster`), cache agent and replication agent and disconnect the application from both databases, run the `ttCWAdmin -stop` command.

```
ttCWAdmin -stop -dsn myDSN
```

Note

If your application connects to the TimesTen database using a virtual IP address, then this connection drops with the `ttCWAdmin -stop` command, since the virtual IP address is managed by Oracle Clusterware. However, if your application connects to the TimesTen database using the host name, the connection is not dropped; however, replication to the standby does not occur.

See Managing Active Standby Pairs in a Cluster and Managing Read-Only Subscribers in the Active Standby Pair for examples showing the sequence in which to use the `ttCWAdmin -start` and `-stop` commands.

Load Cache Groups

If the active standby pair includes cache groups and you have not already loaded the cache group use the `LOAD CACHE GROUP` statement to load the cache group tables from the Oracle database tables.

See Create and Populate a TimesTen Database on One Host.

For more information on when to load cache groups, see Create and Populate a TimesTen Database on One Host.

Include 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

See 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.

Configure 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.

Oracle Clusterware manages the active standby pair, but does not manage the disaster recovery subscriber. The user must explicitly switch to use the remote site 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.

Configure 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` built-in 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.
6. Start the replication agent on the subscriber host.

See Adding or Dropping a Read-Only Subscriber Not Managed by Oracle Clusterware and Rebuilding a Read-Only Subscriber Not Managed by Oracle Clusterware.

Configuring Oracle Clusterware Management with the cluster.oracle.ini File

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

Note

See TimesTen Configuration Attributes for Oracle Clusterware for details on all of the attributes that can be used in the `cluster.oracle.ini` file.

The `ttCWAdmin -create` command reads this file for configuration information, so the location of the text file must be reachable and readable by `ttCWAdmin`. The `ttCWAdmin` utility is used to administer TimesTen active standby pairs in a cluster that is managed by Oracle Clusterware.

It is recommended that you place this file in the TimesTen 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 run the `ttCWAdmin -create` command.

The default location for this file is in the *`timesten_home`*`/conf` directory. 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 system DSN in the `sys.odbc.ini` file. For example, `[basicDSN]` is the entry name in the `cluster.oracle.ini` file described in Configuring Basic Availability. `[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
```

The following sections demonstrate sample configurations of the `cluster.oracle.ini` file:

- Configuring Basic Availability
- Configuring Advanced Availability
- Including Cache Groups in the Active Standby Pair
- Implementing Application Failover
- Configuring for Recovery When Both Master Nodes Permanently Fail

- Using the RepDDL Attribute

Configuring Basic Availability

This example shows an active standby pair with no subscribers.

The host for the active database is the first `MasterHost` defined (`host1`) and the standby database is the second `MasterHost` in the list (`host2`). Each host in the list is delimited by commas. You can include spaces for readability, if desired.

```
[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

Advanced availability involves configuring spare master or subscriber hosts that are idle until needed to replace master or subscriber hosts (used in the active standby pair replication scheme) that either shut down unexpectedly or experience an unrecoverable error.

As mentioned in Configuring Basic Availability, the `MasterHosts` attribute in the `cluster.oracle.ini` file configures the hosts that are used as the master nodes. For an active standby pair replication scheme, you only need two master hosts (one to become the active and one to become the standby). In the event of a failure, the host that did not fail becomes the active (if not already the active) and the failed host is recovered and becomes the standby. However, if the failed host cannot be recovered and if you specified more than two hosts as master hosts in the `cluster.oracle.ini` file, then the next master host in the list can be instantiated to take the place of an unrecoverable master host.

For example, the following shows a configuration of several master hosts. The first two master hosts (`host1` and `host2`) become the active and the standby; the latter two master hosts (`host3` and `host4`) can be used to take the place of either `host1` or `host2` if either encounter an unrecoverable failure.

```
MasterHosts=host1,host2,host3,host4
```

When you configure more than two multiple hosts, you should also configure two virtual IP (VIP) addresses used only by Oracle Clusterware resources that manage TimesTen resources. With these VIP addresses, TimesTen internal processes (those that manage replication) are isolated from any master host changes that may occur because of an unrecoverable host error.

Note

As described in Create the Oracle Clusterware Resources to Manage Virtual IP Addresses, the Oracle Clusterware resource that manage these VIP addresses (used in advanced availability) are created with the `ttCWAdmin -createVIPs` command.

These VIP addresses must be different from any other VIP addresses defined for Oracle Clusterware use or any VIP addresses that are to be used by user applications. Furthermore, if an application does use these VIP addresses, then the application may encounter errors when a master host fails (either recoverable or unrecoverable). These VIP addresses cannot be used by a user application as a method for client failover or as a method to isolate themselves if an active database and standby database switch.

Specify two VIP addresses in the MasterVIP parameter, one for each master host in the active standby pair replication scheme. The VIP addresses specified for the TimesTen cluster must be different from any VIP addresses already defined and used by Oracle Clusterware. In particular, the VIP addresses that are created during the Oracle Clusterware install cannot be used with TimesTen.

```
MasterVIP=192.168.1.1, 192.168.1.2
```

The following parameters are also associated with advanced availability in the `cluster.oracle.ini` file:

- SubscriberHosts, similar to MasterHosts lists the host names that can contain subscriber databases.
- SubscriberVIP, similar to MasterVIP, provides VIP addresses that can be used by TimesTen internally to manage a subscriber node.
- VIPInterface is the name of the public network adaptor.
- VIPNetMask defines the netmask of the virtual IP addresses.

In the following example, the hosts for the active database and the standby database are `host1` and `host2`. The hosts available for instantiation in case of an unrecoverable error are `host3` and `host4`. There are no subscriber nodes. `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
```

The following example configures a single subscriber on `host4`. There is one extra host defined in `SubscriberHosts` that can be used for failover of the master databases and one extra node that can be used for failover of the subscriber database. `MasterVIP` and SubscriberVIP specify the virtual IP addresses defined for the master and subscriber hosts.

```
[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 master nodes:

- Have TimesTen installed
- Have the direct mode application installed if this is part of the configuration. See Implementing Application Failover.

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 sets the CacheConnect attribute to `y`. The 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
```

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.

TimesTen can manage both direct and client/server mode applications that are on the same host as Oracle Clusterware and TimesTen.

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

- 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 running 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`*`]`.

There are also several optional attributes that you can configure, such as AppFailureThreshold, DatabaseFailoverDelay, and AppScriptTimeout. Table A-3 lists and describes all optional attributes and their 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 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.

Note

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
```

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
```

If you set AppType to `DualMaster`, the application starts on both the active and the standby hosts. 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
```

Note

See TimesTen Configuration Attributes for Oracle Clusterware.

Configuring for Recovery When Both Master Nodes Permanently Fail

If both master nodes fail and then come back up, Oracle Clusterware can automatically recover the master databases.

Automatic recovery of a temporary dual failure requires the following:

- 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 the following is true:

- Advanced availability is configured (virtual IP addresses and at least four hosts).
- The active standby pair does not replicate cache groups.
- 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 the active and has become the standby unless the database becomes the active again. Ensure that the shared storage has enough space for two complete database backups. The ttCWAdmin -restore command 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 the following:

- 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. If 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\
```

Monitoring Cluster Status

You can retrieve cluster status and message log files.

The following sections describe how to retrieve the status of the cluster:

- Obtaining Cluster Status
- Message Log Files

Obtaining Cluster Status

The `ttCWAdmin -status` command reports information about all of the active standby pairs in a TimesTen 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.

When you run the `ttCWAdmin -status` command after you have created an active standby pair replication scheme but have not yet started replication, the status appears as follows:

```
% 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
Master Datastore 2:
Host:host2
Status:UNAVAILABLE
State:UNKNOWN
====================================================================
The cluster containing the replicated DSN is offline
```

After you have started the replication scheme and the active database is running but the standby database is not yet running, `ttCWAdmin -status` returns:

```
% 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
Master Datastore 2:
Host:host2
Status:AVAILABLE
State:IDLE
====================================================================
The cluster containing the replicated DSN is online
```

After you have started the replication scheme and the active database and the standby database are both running, `ttCWAdmin -status` returns:

```
% 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
Master Datastore 2:
Host:host2
Status:AVAILABLE
State:STANDBY
====================================================================
The cluster containing the replicated DSN is online
```

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 Error, Warning, and Informational Messages in *Oracle TimesTen In-Memory Database Operations Guide*.

Shutting Down a Cluster

Perform the following to gracefully shut down a cluster:

1. Stop the TimesTen daemon monitor (`ttCRSmaster`), cache agent and replication agent and unload the database (if not in use) with the `ttCWAdmin -stop` command:

   ```
   ttCWAdmin -stop -dsn myDSN
   ```

2. Drop the active standby pair with the `ttCWAdmin -drop` command. This command also deregisters the TimesTen daemon monitor (`ttCRSmaster`) resource from Clusterware.

   ```
   ttCWAdmin -drop -dsn myDSN
   ```

3. Stop the TimesTen cluster agent (`ttCRSAgent`) and TimesTen cluster daemon monitor (`ttCRSDaemon`) on all hosts with the `ttCWAdmin -shutdown` command:

   ```
   ttCWAdmin -shutdown
   ```

> **Note**
>
> By default, the `ttCWAdmin -shutdown` command shuts down the set of hosts defined within the `ttcrsagent.options` file. However, you can specifically identify the hosts you want shut down with the optional `-hosts` argument.
>
> The default behavior is to deregister from Clusterware all TimesTen processes that are registered as Clusterware resources for cluster agents (`ttCRSAgent`) and TimesTen daemon monitors (`ttCRSdaemon`). If the optional `-noderegister` argument is included, TimesTen Clusterware resources will not be deregistered.

4. Prevent the automatic startup of Oracle Clusterware when the server boots by running the Oracle Clusterware `crsctl disable crs` command as root or OS administrator:

   ```
   crsctl disable crs
   ```

5. Optionally, you can gracefully shutdown each TimesTen database on the active, standby and subscriber hosts by disconnecting all applications and then running the following command on each host:

   ```
   ttDaemonAdmin -stop
   ```

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
- Perform a Forced Switchover After Failure of the Active Database or Host

How TimesTen Performs Recovery When Oracle Clusterware is Configured

The TimesTen database monitor (the `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 reconnect 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 a 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. See Automatic Catch-Up of a Failed Master Database.

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.

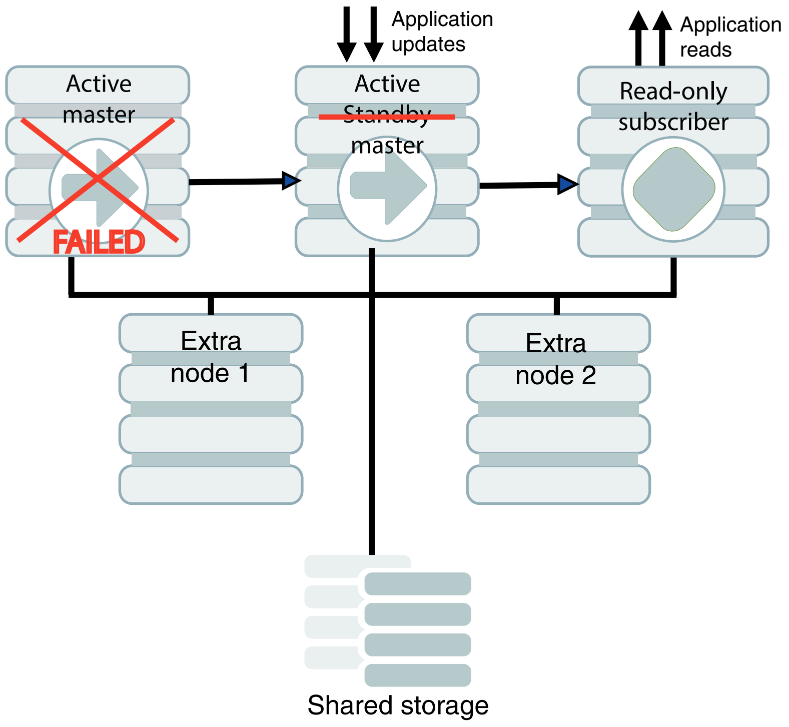
When the conditions for the duplicate operation are satisfied, the existing failed database is destroyed and the duplicate operation starts.

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 8-2 shows that the state of the old standby database has changed to `ACTIVE` and that the application is updating the new active database.

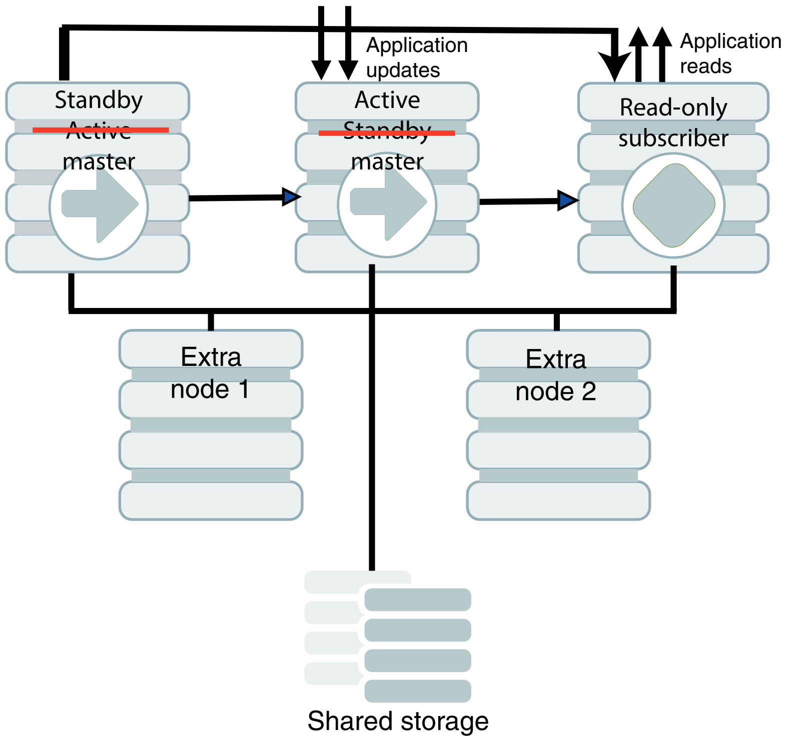
Figure 8-2 Standby Database Becomes Active



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 8-3 shows a cluster where the former active master becomes the standby master.

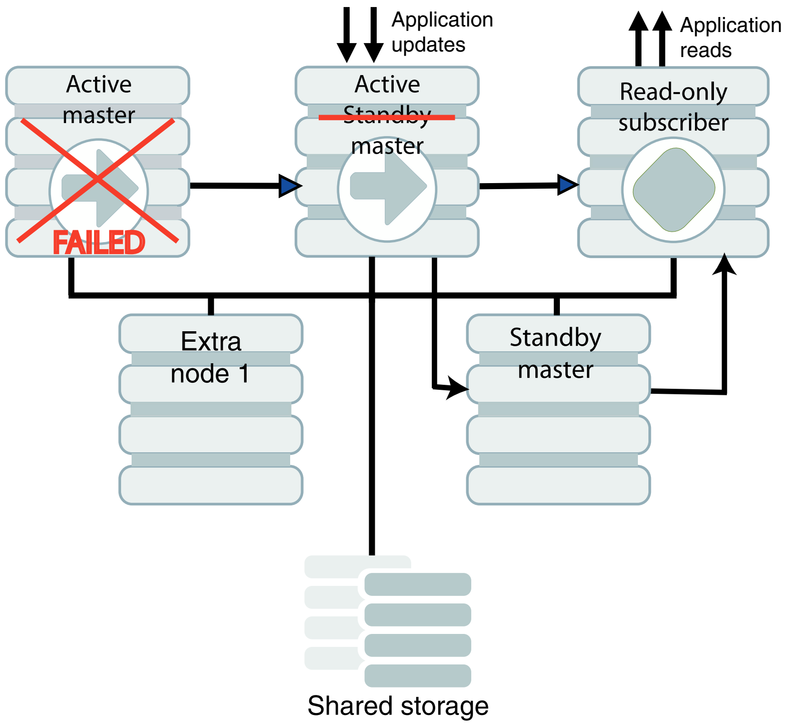
Figure 8-3 Standby Database Starts on Former Active Host



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

Figure 8-4 shows a cluster in which the standby master is started on one of the extra nodes.

Figure 8-4 Standby Database Starts on Extra Host



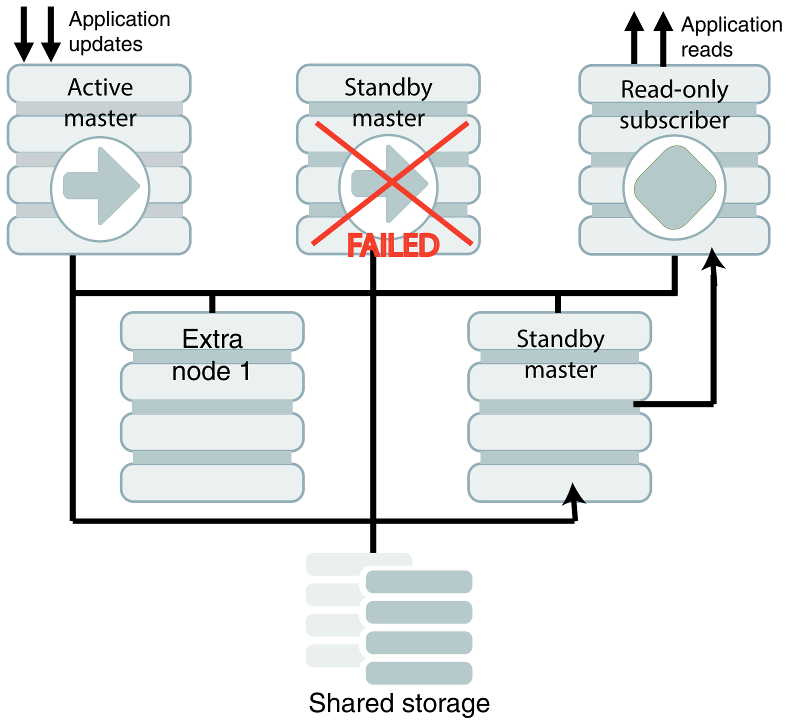
See Perform a Forced Switchover After Failure of the Active Database or Host if you do not want to wait for these automatic actions to occur.

When a Standby Database or Its Host Fails

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

Figure 8-5 shows a cluster in which the standby master is started on one of the extra nodes.

Figure 8-5 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

There are both automatic and manual methods for recovery when failures occur on both master nodes.

This section includes these topics:

- Automatic Recovery
- 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

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

Automatic recovery can occur 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.
- `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 Configuring for Recovery When Both Master Nodes Permanently Fail for examples of `cluster.oracle.ini` files.

Manual Recovery for Advanced Availability

This section assumes that the failed master nodes are recovered to new hosts on which TimesTen and Oracle Clusterware are 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 (`ttCRSAgent`) 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 are recovered to new hosts on which TimesTen and Oracle Clusterware are 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 (`ttCRSAgent`) 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. You can recover to the same master nodes.

To recover to the same master nodes, perform the following steps:

1. Ensure that the TimesTen daemon monitor (`ttCRSmaster`), 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`.

You configure `RETURN TWOSAFE` by using the ReturnServiceAttribute Clusterware attribute in the `cluster.oracle.ini` file.

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 TimesTen daemon monitor (`ttCRSmaster`), replication agent and 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 resides.

   ```
   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 and Manual Recovery for Basic Availability for guidelines on subsequent actions to take.

Perform 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.

Perform the following:

1. Use the `crsctl stop resource` command to stop the TimesTen daemon monitor (`ttCRSmaster`) resource on the active database. This causes the role of the standby database to change to active.

2. Use the `crsctl start resource` command to restart the `ttCRSmaster` resource on the former active database. This causes the database to recover and become the standby database.

The following example demonstrates a forced switchover from the active database on `host1` to the standby database on `host2`.

1. Find all TimesTen resources using the `crsctl status resource` command.

```
% crsctl status resource | grep TT
  NAME=TT_Activeservice_tt181_ttadmin_REP1
  NAME=TT_Agent_tt181_ttadmin_HOST1
  NAME=TT_Agent_tt181_ttadmin_HOST2
  NAME=TT_App_tt181_ttadmin_REP1_updateemp
  NAME=TT_Daemon_tt181_ttadmin_HOST1
  NAME=TT_Daemon_tt181_ttadmin_HOST2
  NAME=TT_Master_tt181_ttadmin_REP1_0
  NAME=TT_Master_tt181_ttadmin_REP1_1
  NAME=TT_Subservice_tt181_ttadmin_REP1
```

2. Find the host where the active database resides by retrieving the status of the `ttCRSActiveService` resource.

```
% crsctl status resource TT_Activeservice_tt181_ttadmin_REP1
  NAME=TT_Activeservice_tt181_ttadmin_REP1
  TYPE=application
  TARGET=ONLINE
  STATE=ONLINE on host1
```

3. There are two `ttCRSmaster` resources listed in the initial status report. Discover which `ttCRSmaster` resource is on the same host as the active database.

```
% crsctl status resource TT_Master_tt181_ttadmin_REP1_0
  NAME=TT_Master_tt181_ttadmin_REP1_0
  TYPE=application
  TARGET=ONLINE
  STATE=ONLINE on host1

% crsctl status resource TT_Master_tt181_ttadmin_REP1_1
  NAME=TT_Master_tt181_ttadmin_REP1_1
  TYPE=application
  TARGET=ONLINE
  STATE=ONLINE on host2
```

4. Stop the `ttCRSmaster` resource on the host where the active database resides.

```
% crsctl stop resource TT_Master_tt181_ttadmin_REP1_0
  CRS-2673: Attempting to stop 'TT_Master_tt181_ttadmin_REP1_0'
  on 'host1'
  CRS-2677: Stop of 'TT_Master_tt181_ttadmin_REP1_0' on
 'host1' succeeded
```

5. Restart the `ttCRSmaster` resource on the former active database.

```
% crsctl start resource TT_Master_tt181_ttadmin_REP1_0
  CRS-2672: Attempting to start 'TT_Master_tt181_ttadmin_REP1_0'
  on 'host1'
  CRS-2676: Start of 'TT_Master_tt181_ttadmin_REP1_0' on
 'host1' succeeded
```

6. Confirm that the forced switchover succeeds by checking where the active service `ttCRSActiveService` and standby service `ttCRSsubservice` resources are located.

```
% crsctl status resource TT_Activeservice_tt181_ttadmin_REP1
  NAME=TT_Activeservice_tt181_ttadmin_REP1
  TYPE=application
  TARGET=ONLINE
  STATE=ONLINE on host2

% crsctl status resource TT_Subservice_tt181_ttadmin_REP1
  NAME=TT_Subservice_tt181_ttadmin_REP1
  TYPE=application
  TARGET=ONLINE
  STATE=ONLINE on host1
```

See the Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation for more information about the `crsctl start resource` and `crsctl stop resource` commands.

Clusterware Management

There are certain procedures for managing clusterware when used in conjunction with TimesTen.

This section includes the following topics:

- Changing User Names or Passwords When Using Oracle Clusterware
- Managing Hosts in the Cluster
- Managing Active Standby Pairs in a Cluster
- Managing Read-Only Subscribers in the Active Standby Pair
- Reversing the Roles of the Master Databases
- Modifying Connection Attribute Values
- Managing the TimesTen Database RAM Policy
- Changing the Schema
- Making Schema Changes to Cache Groups
- Moving a Database to a Different Host
- Performing a Rolling Upgrade of Oracle Clusterware Software
- Upgrading TimesTen When Using Oracle Clusterware
- Performing Host or Network Maintenance

Changing User Names or Passwords When Using Oracle Clusterware

When you create the active standby pair replication scheme with the `ttCWAdmin -create` command, Oracle Clusterware prompts for the required user names and passwords in order to manage the TimesTen environment.

Oracle Clusterware stores these user names and passwords. After modifying any user name or password, you must run the `ttCWAdmin -reauthenticate` command to enable Oracle Clusterware to store these new user names and passwords.

1. Ensure that the `DDLReplicationLevel` connection attribute is set to 3. This value replicates changes to the user names or passwords on the active database to the standby database.
2. Modify any of the user names or passwords in the same manner (and with the same restrictions) as described in Changing User Names or Passwords Used by Replication.

3. Ensure that all password changes are replicated to the standby database by calling the `ttRepSubscriberWait` built-in procedure (or the `ttRepAdmin -wait` command) on the active database using the DSN and host of the standby database. For example, to ensure that all transactions are replicated to the `master2` standby database on the `host2` host:

   ```
   CALL ttRepSubscriberWait(NULL, NULL, 'master2', 'host2', -1);
   ```

4. Store the new passwords in Oracle Clusterware by running the `ttCWAdmin -reauthenticate` command.

   ```
   ttCWAdmin -reauthenticate -dsn myDSN
   ```

   This command prompts for the same information as requested for the `ttCWAdmin -create` command, which is discussed in Create an Active Standby Pair Replication Scheme.

Managing Hosts in the Cluster

The following sections describe how to add or remove hosts when using a cluster.

- Adding a Host to the Cluster
- Removing a Host from the Cluster

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
```

Managing Active Standby Pairs in a Cluster

The following sections describe how to add or remove an active standby pair to a cluster.

- Adding an Active Standby Pair to a Cluster

- Removing an Active Standby Pair from a Cluster

Adding an Active Standby Pair to a Cluster

You can add an active standby pair (with or without subscribers) to a cluster that is already managing an active standby pair.

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.
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 as the `root` user.

   ```
   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
   ```

Removing an Active Standby Pair from a Cluster

You can remove an active standby pair (with or without subscribers) from a cluster.

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.

   ```
   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
```

See ttDestroy in *Oracle TimesTen In-Memory Database Reference*.

Managing Read-Only Subscribers in the Active Standby Pair

The following sections describe how to manage read-only subscribers in the active standby pair that is managed by Oracle Clusterware.

- Adding a Read-Only Subscriber Managed by Oracle Clusterware
- Removing a Read-Only Subscriber Managed by Oracle Clusterware
- Adding or Dropping a Read-Only Subscriber Not Managed by Oracle Clusterware
- Rebuilding a Read-Only Subscriber Not Managed by Oracle Clusterware

Adding a Read-Only Subscriber Managed by Oracle Clusterware

To add a read-only subscriber that is to be managed by Oracle Clusterware to an active standby pair replication scheme, 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 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 Managed by Oracle Clusterware

To remove a read-only subscriber that is managed by Oracle Clusterware 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 or Dropping a Read-Only Subscriber Not Managed by Oracle Clusterware

You can add or drop a read-only subscriber that is not managed by Oracle Clusterware to or from an existing active standby pair replication scheme that is managed by Oracle Clusterware.

Using the `ttCWAdmin -beginAlterSchema` command enables you to add a subscriber without dropping and re-creating 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. Run the `ttCWAdmin -beginAlterSchema` command to stop the replication agent on the active and standby databases.
2. Using `ttIsql` to connect to the active database, you can add or drop the subscriber to or from the replication scheme by using an `ALTER ACTIVE STANDBY PAIR` statement. For example, to add a subscriber:

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

   To drop a subscriber:

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

3. Run the `ttCWAdmin -endAlterSchema` command that registers the altered replication scheme and starts replication. If you are adding a subscriber, this also initiates a duplicate to the standby database.
4. Run the `ttIsql repschemes` command to verify that the read-only subscriber has been added to or dropped from the replication scheme.
5. Use the `ttRepStateGet` built-in procedure to verify that the state of the standby database is `STANDBY`.
6. If you added a subscriber, then run `ttRepAdmin -duplicate` on the subscriber host to duplicate the standby database to the read-only subscriber. See Duplicating a Database.
7. If you added a subscriber, start the replication agent on the subscriber host.

If you added a subscriber, ensure that the read-only subscriber is included if the cluster is dropped and re-created by adding the RemoteSubscriberHosts Oracle Clusterware attribute for the read-only subscriber in the `cluster.oracle.ini` file as described in Step 1 in Configure a Read-Only Subscriber That Is Not Managed by Oracle Clusterware. Alternatively, if you dropped a subscriber, remove the `RemoteSubscriberHosts` Oracle Clusterware attribute for the dropped subscriber in the `cluster.oracle.ini` file (if it is configured).

Rebuilding a Read-Only Subscriber Not Managed by Oracle Clusterware

Perform the following tasks to destroy and rebuild a read-only subscriber that is not managed by Oracle Clusterware:

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.

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.

Optionally, you can also use the `-timeout` option with the `-switch` option to set a timeout for the number of seconds to wait for the active and standby database switch to complete.

For example:

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

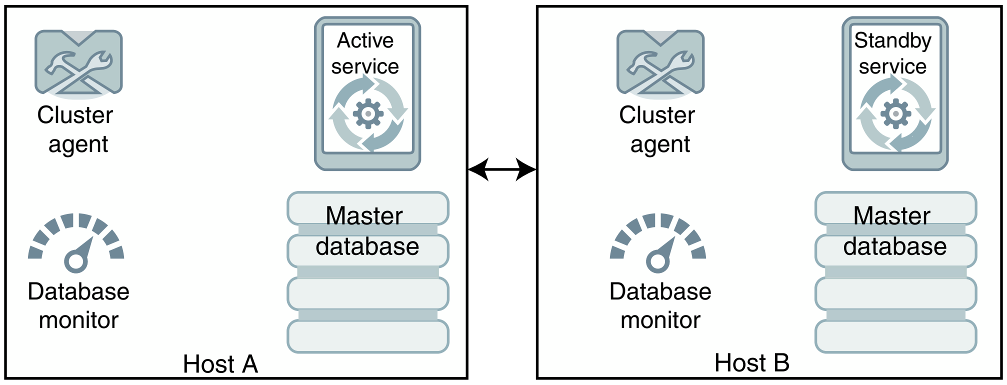
Ensure that there are no open transactions before using the `-switch` option. If there are open transactions, the command fails.

Note

See ttCWAdmin in the *Oracle TimesTen In-Memory Database Reference*.

Figure 8-6 shows the hosts for an active standby pair. The active database resides on host A, and the standby database resides on host B.

Figure 8-6 Hosts for an Active Standby Pair



The `ttCWAdmin -switch` command performs these tasks:

- Deactivates the TimesTen cluster agent (`ttCRSAgent`) on host A (the active master).
- Disables the TimesTen 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.

Modifying Connection Attribute Values

When you modify connection attributes across an active standby pair with subscribers, the connection attributes must be modified on all hosts within this configuration.

Note

You cannot modify any `DATASTORE` connection attributes since they are only allowed to be set at data store creation time. For example, this procedure can be used to change the `PermSize` value.

Use the `ttCWAdmin -beginAlterSchema` and `-endAlterSchema` commands to facilitate the change of any connection attribute values on the active and standby databases and any subscribers.

- The `ttCWAdmin -beginAlterSchema` command suspends the Oracle Clusterware management and stops the replication agents on the active and standby databases and any subscriber databases in preparation for any changes.
- After you complete all changes, the `ttCWAdmin -endAlterSchema` command resumes Oracle Clusterware management and restarts all replication agents on the active and standby databases and any subscriber databases.

Perform the following tasks when altering any connection attributes for the active standby pair when using Oracle Clusterware:

1. Suspend Oracle Clusterware and stop all replication agents for the active and standby databases with the `ttCWAdmin -beginAlterSchema` command.

   The active database continues to accept requests and updates, but any changes are not propagated to the standby database and any subscribers until the replication agents are restarted.

   The `ttCWAdmin -beginAlterSchema` command also changes the RAM policy temporarily for the standby database and all subscriber databases to *InUse with RamGrace* where the grace period is set for 60 seconds to enable these databases to be unloaded by TimesTen. Once the standby and subscriber databases are unloaded from memory, the connection attributes for these databases can be modified.

   ```
   ttCWAdmin -beginAlterSchema -dsn advancedDSN
   ```

2. Disconnect any application connections and wait for the standby and subscriber databases to unload from memory (based on the RAM policy).

   Once the standby and subscriber databases are unloaded from memory, alter any connection attributes, such as `PermSize`, on the hosts for the standby and all subscriber databases in their respective `sys.odbc.ini` files.

3. Resume Oracle Clusterware and restart all replication agents for the active and standby databases with the `ttCWAdmin -endAlterSchema` command. The configured RAM policy for each TimesTen database is set back to *always*. The active database propagates any transactions that occurred while the standby database and subscribers were down.

   ```
   ttCWAdmin -endAlterSchema -dsn advancedDSN
   ```

   > **Note**
   >
   > Wait an appropriate amount of time for all changes to propagate from the active database to the standby database and all subscribers before performing the next step.
   >
   > The only host that has not had the connection attribute change is the active database. You will switch the active database with the standby database so that you can modify the connection attributes on this host.

4. Suspend all application workload and disconnect all applications on the active database.

5. Switch the active and standby databases with the `ttCWAdmin -switch` command.

   ```
   ttCWAdmin -switch -dsn advancedDSN
   ```

   > **Note**
   >
   > See Reversing the Roles of the Master Databases.

6. Suspend Oracle Clusterware and stop all replication agents for all databases with the `ttCWAdmin -beginAlterSchema` command.

   The new active database may still accept requests and updates, but any changes are not propagated to the new standby database and any subscribers.

   The RAM policy changes for the new standby database (and all subscriber databases) to *inUse with RamGrace* where the grace period is set for 60 seconds to enable these databases to be unloaded by TimesTen.

   ```
   ttCWAdmin -beginAlterSchema -dsn advancedDSN
   ```

7. Wait for the new standby database to unload from memory. Once unloaded, alter the same connection attributes, such as `PermSize`, on the new standby database in its `sys.odbc.ini` file. The connection attributes are now modified on all hosts.

8. Run the `ttCWAdmin -endAlterSchema` command to resume Oracle Clusterware management and restart the replication agents on the active and standby databases. The configured RAM policy resumes to *always*.

   ```
   ttCWAdmin -endAlterSchema -dsn advancedDSN
   ```

9. Suspend all application workload and disconnect all applications on the active database.

10. If desired, you can switch the active and standby databases with the `ttCWAdmin -switch` command to restore the active standby pair to the original configuration.

```
ttCWAdmin -switch -dsn advancedDSN
```

Managing the TimesTen Database RAM Policy

By default, the TimesTen database RAM policy is set to *always* when Oracle Clusterware manages the TimesTen database. However, if you stop Oracle Clusterware management, the TimesTen database RAM policy is set to *inUse*.

If you no longer use Oracle Clusterware to manage TimesTen, you should set the TimesTen RAM policy to what is appropriate for your environment. Typically, the recommended setting is *manual*.

See Specifying a RAM Policy in the *Oracle TimesTen In-Memory Database Operations Guide*.

Changing the Schema

When using Oracle Clusterware to manage an active standby pair, you can modify the schema by running DDL statements as in a normal replication environment, except that Oracle Clusterware must start and stop all replication agents, when it is necessary to do so.

Thus, when you change the schema, note the following:

- For those DDL statements on objects that are automatically replicated, you do not need to stop the replication agents. In this case, no further action is required, since these DDL statements are automatically propagated and applied to the standby database and any subscribers. The `DDLReplicationLevel` connection attribute controls what DDL statements are replicated.
- For those objects that are a part of the replication scheme, but any DDL statements processed on these objects are not replicated (these objects are listed in Making Other Changes to an Active Standby Pair), run the Oracle Clusterware `ttCWAdmin -beginAlterSchema` command on the active database, which suspends any Oracle Clusterware management and stops the replication agents on each node in the replication scheme. Then, run the DDL statement on the active database in the replication scheme. Finally, run the Oracle Clusterware `ttCWAdmin -endAlterSchema` command on the active database to restart all replication agents.

  Because these objects are a part of the replication scheme, but the DDL statements are not replicated, a duplicate occurs after the `ttCWAdmin -endAlterSchema` command to propagate these schema changes to the standby database and any subscribers. This is the only scenario when a duplicate is used to propagate the schema changes.

  Follow the instructions described in Facilitating Schema Change for Oracle Clusterware.
- For those DDL statements on objects that are not automatically replicated and are not part of the replication scheme, run the Oracle Clusterware `ttCWAdmin -beginAlterSchema` command on the active database, which suspends any Oracle Clusterware management and stops and the replication agents on all nodes. Then, you can synchronize all nodes by manually running these DDL statements as indicated in Making DDL Changes in an Active Standby Pair. Finally, run the Oracle Clusterware `ttCWAdmin -endAlterSchema` command on the active database to restart all replication agents.

  Follow the instructions described in Facilitating Schema Change for Oracle Clusterware.

Note

The Making DDL Changes in an Active Standby Pair and Making Other Changes to an Active Standby Pair sections describe which DDL statements are and are not automatically replicated for an active standby pair. These sections also describe what objects are a part of the replication scheme.

Facilitating Schema Change for Oracle Clusterware

Use the `ttCWAdmin -beginAlterSchema` and `-endAlterSchema` commands to facilitate a schema change on the active and standby databases.

- The `ttCWAdmin -beginAlterSchema` command suspends the Oracle Clusterware management and stops replication agents on both the active and standby databases in preparation for any schema changes.
- After you complete all schema changes, run the `ttCWAdmin -endAlterSchema` command. For those objects that are a part of the replication scheme, but any DDL statements processed on these objects are not automatically replicated, a duplicate occurs after the `ttCWAdmin -endAlterSchema` command to propagate only these schema changes to the standby database and any subscribers. This command registers the altered replication scheme, restarts the replication agents on the active and standby databases, and reinstates Oracle Clusterware control.

Perform the following tasks when altering the schema of the active standby pair when using Oracle Clusterware:

1. Suspend Oracle Clusterware and stop the replication agents on both the active and standby databases.

   ```
   ttCWAdmin -beginAlterSchema -dsn advancedDSN
   ```

2. Make any desired schema changes.

   If you create, alter, or drop any objects where the DDL for these objects are not replicated, you should also manually create, alter, or drop the same objects on the standby database and subscribers while the replication agents are inactive to ensure that the same objects exist on all databases in the replication scheme. For example, if you create a materialized view on the active database, create the materialized view on the standby and subscriber databases at this time.

3. If the object is not automatically replicated but is a part of the replication scheme, (such as 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. If the object is a cache group, see Making Schema Changes to Cache Groups for instructions to create, alter, or drop a cache group.

5. Run the `ttCWAdmin -endAlterSchema` command to resume Oracle Clusterware and restart the replication agents on the active and standby databases. If you modified objects that are a part of the replication scheme, but any DDL statements processed on these objects are not automatically replicated, a duplicate occurs after the `ttCWAdmin -endAlterSchema` command to propagate only these schema changes to the standby database and any subscribers.

   ```
   ttCWAdmin -endAlterSchema -dsn advancedDSN
   ```

Making Schema Changes to Cache Groups

Use the `ttCWAdmin -beginAlterSchema` and `-endAlterSchema` commands to facilitate the schema changes on cache groups within Oracle Clusterware.

- Add a Cache Group
- Drop a Cache Group
- Change an Existing Cache Group

Add a Cache Group

You can add a cache group on the active database of the active standby pair.

Perform these steps on the active database of the active standby pair.

1. Suspend Oracle Clusterware management and stop the replication agents with the `ttCWAdmin -beginAlterSchema` command.

   ```
   ttCWAdmin -beginAlterSchema -dsn advancedDSN
   ```

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. Resume Oracle Clusterware and start the replication agents by running the `ttCWAdmin -endAlterSchema` command. Since you added a cache group object, a duplicate occurs to propagate these schema changes to the standby database.

   ```
   ttCWAdmin -endAlterSchema -dsn advancedDSN
   ```

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

Drop a Cache Group

Dropping a cache group within a Clusterware environment requires several steps.

Perform these steps to drop a cache group.

1. Unload the cache group.

   ```
   UNLOAD CACHE GROUP samplecachegroup;
   ```

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

   ```
   ttCWAdmin -beginAlterSchema -dsn advancedDSN
   ```

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_home*/install/ oraclescripts/cacheCleanUp.sql SQL*Plus script as the cache administration user on the Oracle database to drop the Oracle database objects used to implement autorefresh operations.
7. Resume Oracle Clusterware and restart the replication agents by running the ttCWAdmin -endAlterSchema command. Since you dropped a cache group object, a duplicate occurs to propagate these schema changes to the standby database.

   ```
   ttCWAdmin -endAlterSchema -dsn advancedDSN
   ```

Change an Existing Cache Group

Changing an existing cache group involves dropping and adding the cache group.

To change an existing cache group, first drop the existing cache group as described in Drop a Cache Group. Then add the cache group with the desired changes as described in Add a Cache Group.

Moving a Database to a Different Host

When a cluster is configured for advanced availability, you can use the ttCWAdmin -relocate command 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 database and the old standby database becomes the new active database.

The ttCWAdmin -relocate command 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.

If the ttCWAdmin -relocate command requires a role switch, then you can optionally use the -timeout option with the -relocate option to set a timeout for the number of seconds to wait for the role switch.

For example:

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

Note

See ttCWAdmin in the *Oracle TimesTen In-Memory Database Reference*.

Performing a Rolling Upgrade of Oracle Clusterware Software

There are methods you can run to perform a rolling upgrade of the Oracle Clusterware software.

See Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation.

Upgrading TimesTen When Using Oracle Clusterware

There are methods to upgrade TimesTen on all hosts when using Oracle Clusterware:

- Performing an Offline TimesTen Upgrade When Using Oracle Clusterware in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.
- Performing an Online TimesTen Upgrade When Using Oracle Clusterware in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Performing Host or Network Maintenance

When you need to perform host or network maintenance, you need to stop the Oracle Clusterware resources and take down one or more of the TimesTen databases in the cluster.

In order to maintain data consistency for the database, you need to ensure that the TimesTen databases included in the active standby pair are brought down properly so that no transactions are lost.

One of the decisions you will make during performing maintenance is whether you should leave Oracle Clusterware enabled or disabled. If you leave Oracle Clusterware enabled, then all Oracle Clusterware and TimesTen processes restart automatically after a system reboot. If you disable Oracle Clusterware, none of these processes restart automatically.

- Perform Maintenance on All Hosts in the Cluster Simultaneously
- Perform Maintenance While Still Accepting Requests

Perform Maintenance on All Hosts in the Cluster Simultaneously

You can perform tasks to facilitate minimal down time while performing maintenance on all hosts in the cluster.

Note

If you have an active, standby and one or more subscriber databases, you need to run some of these tasks on each host that contains the designated database.

1. Stop Oracle Clusterware and the replication agents by running the Oracle Clusterware `crsctl stop crs` command as `root` or OS administrator on each of the hosts that contain the active, standby, and subscriber databases.

   Since the active database is down, all requests are refused until the replication agents are restarted.

   ```
   crsctl stop crs
   ```

   The `crsctl stop crs` command changes the RAM policy temporarily for the active, standby, and all subscriber databases to *inUse with RamGrace* where the grace period is set for 60 seconds to enable these databases to be unloaded by TimesTen.

2. Optionally, you can prevent the automatic startup of Oracle Clusterware and TimesTen when the server boots by running the Oracle Clusterware `crsctl disable crs` command as `root` or OS administrator on all hosts in the cluster:

   ```
   crsctl disable crs
   ```

3. Disconnect any application connections and wait for the active, standby, and subscriber databases to unload from memory.

4. To gracefully shutdown each TimesTen database in the active standby pair, run the following command on each of the hosts that contain the active, standby, and subscriber databases:

   ```
   ttDaemonAdmin -stop
   ```

   > **Note**
   >
   > The TimesTen main daemon process manages all databases under the same TimesTen installation, be sure to disconnect from all databases before running the above command.

5. Perform maintenance on the hosts that contain the active, standby, and subscriber databases.
6. After the maintenance is complete, either:
   - Reboot all hosts, then wait until the Oracle Clusterware and TimesTen processes are running (which can take several minutes) if you did not disable the automatic startup of Oracle Clusterware and TimesTen.
   - Perform the following tasks on each host in the cluster if you disabled the automatic startup of Oracle Clusterware and TimesTen after a reboot or if you are not rebooting the hosts after maintenance when automatic startup is enabled:
     a. Start the TimesTen database by running the following command:

        ```
        ttDaemonAdmin -start
        ```

     b. Enable the automatic startup of Oracle Clusterware when the server boots by running `crsctl enable crs` as `root` or OS administrator:

        ```
        crsctl enable crs
        ```

     c. Start Oracle Clusterware on the local server by running `crsctl start crs` as `root` or OS administrator. Wait until all of the Oracle Clusterware resources come up before continuing to the next step.

        ```
        crsctl start crs
        ```

   Once everything is up, you can reconnect with your applications and the active starts to replicate all updates to the standby and subscriber databases. The configured RAM policy resumes to *always*.

Perform Maintenance While Still Accepting Requests

There are methods to provide minimal down time while performing maintenance on all hosts in the cluster.

> **Note**
>
> If you have an active, standby and one or more subscriber databases, you need to run some of these tasks on each host that contains the designated database.

1. Stop Oracle Clusterware and the replication agents by running the Oracle Clusterware `crsctl stop crs` command as `root` or OS administrator on each of the hosts that contain the standby and subscriber databases.

The active database continues to accept requests and updates, but any changes are not propagated to the standby database and any subscribers until the replication agents are restarted.

```
crsctl stop crs
```

The `crsctl stop crs` command also changes the RAM policy temporarily for the standby and all subscriber databases to *inUse with RamGrace* where the grace period is set for 60 seconds to enable these databases to be unloaded by TimesTen.

2. Optionally, you can prevent the automatic startup of Oracle Clusterware and TimesTen when the server boots by running the Oracle Clusterware `crsctl disable crs` command as `root` or OS administrator on each of the hosts that contain the standby and subscriber databases.

   ```
   crsctl disable crs
   ```

3. Disconnect any application connections and wait for the standby and subscriber databases to unload from memory.
4. To gracefully shutdown a TimesTen database, run the following command on each of the hosts that contain the standby and subscriber databases:

   ```
   ttDaemonAdmin -stop
   ```

   Note

   The TimesTen main daemon process manages all databases under the same TimesTen installation, be sure to disconnect from all databases before running the above command.

5. Perform maintenance on the hosts that contain the standby and subscriber databases.
6. After the maintenance is complete, either:
   - Reboot all hosts, then wait until the Oracle Clusterware and TimesTen processes are running (which can take several minutes) if you did not disable the automatic startup of Oracle Clusterware and TimesTen.
   - Perform the following tasks on each host in the cluster if you disabled the automatic startup of Oracle Clusterware and TimesTen after a reboot or if you are not rebooting the hosts after maintenance when automatic startup is enabled:
     a. Start the TimesTen database by running the following command:

        ```
        ttDaemonAdmin -start
        ```

     b. Enable the automatic startup of Oracle Clusterware when the server boots by running `crsctl enable crs` as `root` or OS administrator:

        ```
        crsctl enable crs
        ```

     c. Start Oracle Clusterware on the local server by running `crsctl start crs` as `root` or OS administrator. Wait until all of the Oracle Clusterware resources come up before continuing to the next step.

        ```
        crsctl start crs
        ```

   Once everything is up, the active replicates all updates to the standby and subscriber databases.
7. Switch the active and standby databases with the `ttCWAdmin -switch` command so you can perform the same maintenance on the host with the active database.

```
ttCWAdmin -switch -dsn advancedDSN
```

> **Note**
>
> See Reversing the Roles of the Master Databases for more details on the `ttCWAdmin -switch` command.

8. Stop Oracle Clusterware and the replication agents by running the Oracle Clusterware `crsctl stop crs` command as `root` or OS administrator on the host with the new standby database.

   The new active database continues to accept requests and updates, but any changes are not propagated to the new standby database and any subscribers until the replication agents are restarted.

   ```
   crsctl stop crs
   ```

9. Disconnect any application connections and wait for the standby and subscriber databases to unload from memory.
10. To gracefully shutdown the TimesTen database, run the following command on the host that contains the new standby database:

    ```
    ttDaemonAdmin -stop
    ```

11. Perform maintenance on the host that contains the new standby database. Now the maintenance has been performed on all hosts in the cluster.
12. After the maintenance is complete, either:
    - Reboot all hosts, then wait until the Oracle Clusterware and TimesTen processes are running (which can take several minutes) if you did not disable the automatic startup of Oracle Clusterware and TimesTen.
    - Perform the following tasks on each host in the cluster if you disabled the automatic startup of Oracle Clusterware and TimesTen after a reboot or if you are not rebooting the hosts after maintenance when automatic startup is enabled:
      a. Start the TimesTen database by running the following command:

         ```
         ttDaemonAdmin -start
         ```

      b. Enable the automatic startup of Oracle Clusterware when the server boots by running `crsctl enable crs` as `root` or OS administrator:

         ```
         crsctl enable crs
         ```

      c. Start Oracle Clusterware on the local server by running `crsctl start crs` as `root` or OS administrator. Wait until all of the Oracle Clusterware resources come up before continuing to the next step.

         ```
         crsctl start crs
         ```

    Once everything is up, the active replicates all updates to the standby and subscriber databases. The RAM policy resumes to *always*.

13. Switch back to the original configuration for the active and standby roles for the active standby pair with the `ttCWAdmin -switch` command.

    ```
    ttCWAdmin -switch -dsn advancedDSN
    ```

Note

See the Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation.

9

Defining Classic Replication Schemes

This chapter describes how to define classic replication schemes.

Note

For information about defining active standby pair replication schemes, see Defining an Active Standby Pair Replication Scheme. If you want to replicate a database that has cache groups, see Administering an Active Standby Pair with Cache Groups.

This chapter includes these topics:

Note

To reduce the amount of bandwidth required for replication, see Compressing Replicated Traffic.

To replicate tables with columns in a different order or with a different number of partitions, see Column Definition Options for Replicated Tables.

Designing a Highly Available System

There are several 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.

You can configure the following classic replication schemes (as described in Types of Replication Schemes):

- 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 enable 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

See Performing an Online Upgrade with Classic Replication in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Considering Failover and Recovery Scenarios

As you plan a classic 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 Managing Database Failover and Recovery.

When planning failover strategies, consider which subscribers are to 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.

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 in a Classic Replication Scheme.

Figure 9-1 Recovering a Master in a Unidirectional Scheme

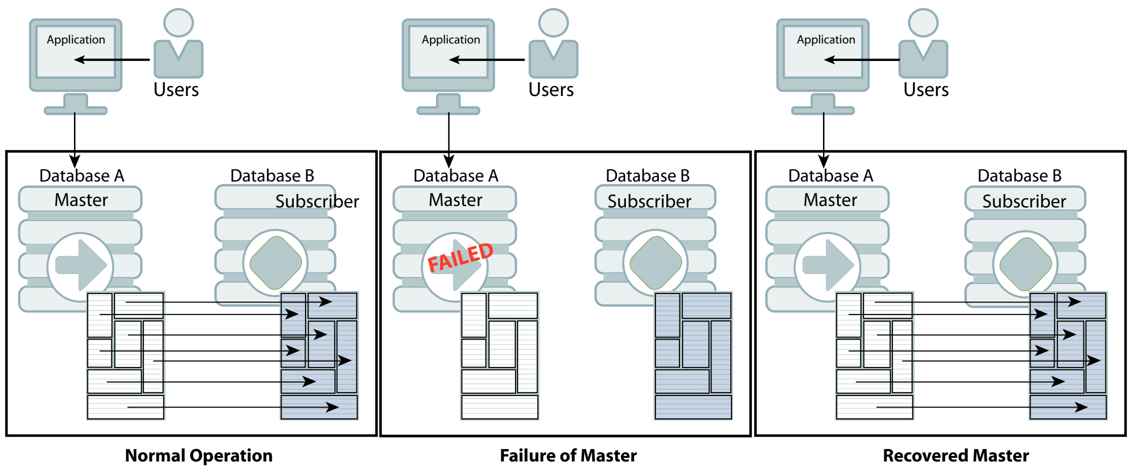
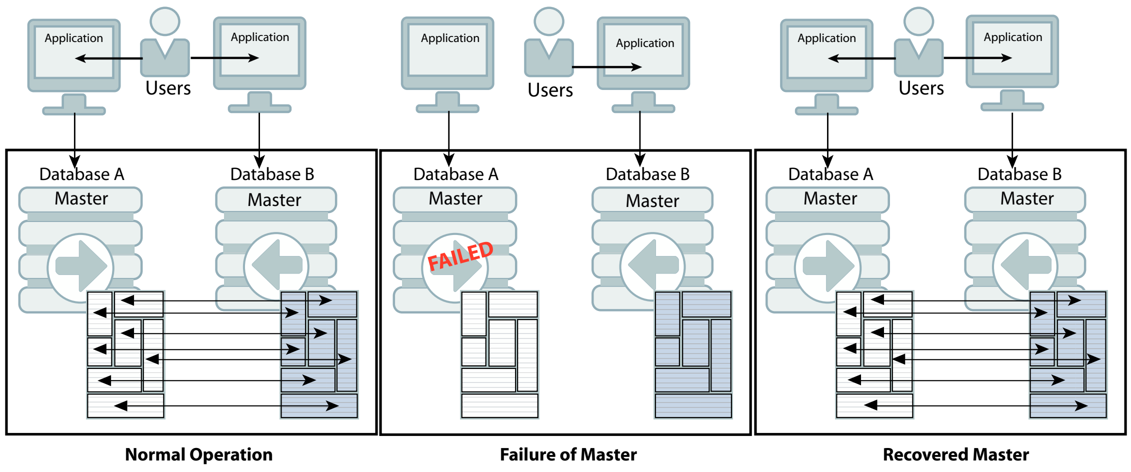


Figure 9-2 shows a bidirectional distributed workload scheme in which the entire database is replicated. Failover in this type of classic replication scheme involves shifting the users of the application on the failed database to the application on the surviving database. Upon recovery, the workload 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 in a Classic Replication Scheme. Upon recovery, the users can be moved back to the recovered master database.

Propagation classic 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 classic 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 classic 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.

Table 9-1 (Cont.) Performance and Recovery Tradeoffs

Type of Behavior	Asynchronous Replication (Default)	Return Receipt	Return Twosafe
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 enables 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.

See Using a Return Service in a Classic Replication Scheme.

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 Resolving Replication Conflicts.

Defining a Classic Replication Scheme

After you have designed a classic 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. See CREATE REPLICATION in the *Oracle TimesTen In-Memory Database SQL Reference*.

Table 9-2 Components of a Replication Scheme

Component	See...
`CREATE REPLICATION` *`Owner.SchemeName`*	Owner of the Classic Replication Scheme and Replicated Objects
`ELEMENT` *`ElementName ElementType`*	Defining Replication Elements
`[`*`CheckConflicts`*`]`	Checking for Replication Conflicts on Table Elements
`{MASTER|PROPAGATOR}` *`DatabaseName`* `ON "`*`HostName`*`"`	Database Names
`[TRANSMIT {NONDURABLE|DURABLE}]`	Setting Transmit Durability on DATASTORE Element
`SUBSCRIBER` *`DatabaseName`* `ON "`*`HostName`*`"`	Database Names
`[`*`ReturnServiceAttribute`*`]`	Using a Return Service in a Classic Replication Scheme
`INCLUDE|EXCLUDE`	Defining the DATASTORE Element
`STORE` *`DatabaseName DataStoreAttributes`*	Setting STORE Attributes in a Classic Replication Scheme
`[`*`NetworkOperation`*`]`	Configuring Network Operations for a Classic Replication Scheme

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 classic replication scheme used by a database persists across system reboots. Modify a classic replication scheme by using the `ALTER REPLICATION` statement. See Altering a Classic Replication Scheme.

Owner of the Classic Replication Scheme and Replicated Objects

The classic 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

There are three roles of the databases in a classic 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 classic replication scheme, you need to define the data source names (DSNs) for the databases in the replication scheme. On UNIX or Linux platforms, create an `odbc.ini` file. See Configuring a Classic Replication Scheme with One Master and One Subscriber for an example.

Each database "name" specified in a classic 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
```

Restrictions and Table Requirements for Classic Replication Schemes

All masters and subscribers must have their clocks synchronized through NTP or other means. The clock skew between all masters and subscribers cannot exceed 250 milliseconds. When adjusting the system clocks on any nodes to be synchronized with each other, do not set any clock backward in time.

The name and owner of replicated tables participating in the classic replication scheme must be identical on the master and subscriber databases. However, the definition for the columns of replicated tables participating in the replication scheme do not necessarily need to be identical. For more information on the column definition options, see Column Definition Options for Replicated Tables.

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 restrictions:

- 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*.

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 for the classic replication scheme, 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.

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.

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 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.

Note

Enable the `Preallocate` attribute to ensure that there is sufficient space for the new database. See Preallocate in the *Oracle TimesTen In-Memory Database Reference*.

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 Classic Replication Scheme or Creating and Adding a Subscriber Database to a Classic Replication Scheme.
2. Apply the replication scheme to the master database. See Applying a Classic Replication Scheme to a Database.
3. Start the replication agent for the master database. See Starting and Stopping the Replication Agents.
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 Classic chooses them.
   >
   > See ttRepAdmin in *Oracle TimesTen In-Memory Database Reference*.

6. Start the replication agent on the subscriber database.

Restrictions for Classic Replication Schemes Involving Multiple Masters

Designing bidirectional replication schemes are a commonly used design for classic replication. The original design for bidirectional replication was to include only two masters. However, you are not restricted in limiting your design to only two masters in your bidirectional replication design.

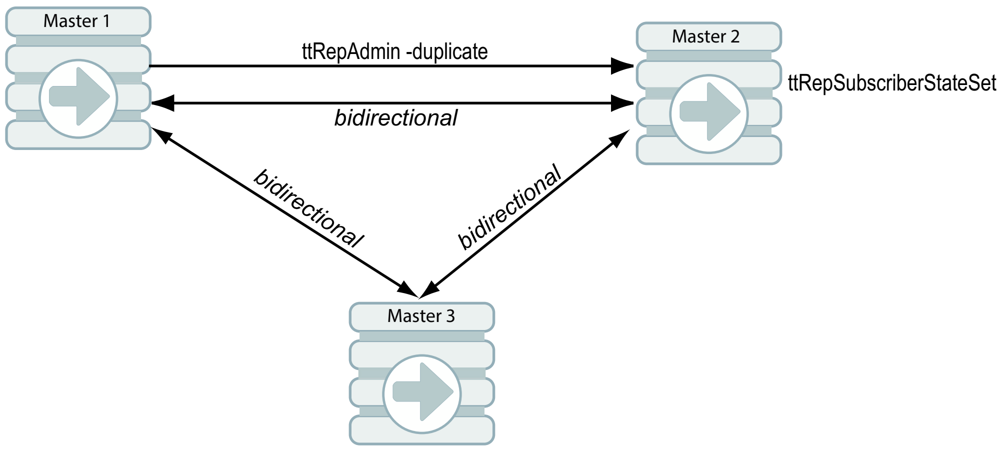
If you decide to use more than two masters (a multi-master topology) and if you decide to use `ttRepAdmin -duplicate` to duplicate another store, you must reset the replication states for the duplicated stores subscribers with the `ttRepSubscriberStateSet` built-in procedure to set all subscribers to the appropriate state.

Note

See Set the Replication State of Subscribers.

As shown in Figure 9-3, you have three masters (`master1`, `master2`, and `master3`) each configured with a bidirectional replication scheme with each other. If you decide to re-create `master2` from `master1` by running `ttRepAdmin -duplicate` on `master1`, then you must call the `ttRepSubscriberStateSet` built-in procedure on `master2` to set the replication states for `master3`.

Figure 9-3 Multiple Masters Involved in Bidirectional Replication Scheme



Defining Replication Elements

A classic 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.

Note

If you want to replicate a database with cache groups, see 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 Classic Replication Schemes. The correct way to propagate elements is described in Propagation Scheme.

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 Classic Replication Scheme. 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 Classic Replication Scheme.

The rest of this section includes the following topics:

- Defining the DATASTORE Element
- Defining Table Elements
- Replicating Tables with Foreign Key Relationships in a Classic Replication Scheme
- Replicating Sequences
- Views and Materialized Views in a Replicated Database

Defining the DATASTORE Element

You can replicate the entire contents of a master database to a subscriber by 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.

Note

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

You can replicate tables from a master database to a subscriber database.

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 classic replication schemes, see Restrictions and Table Requirements for Classic Replication Schemes.

Replicating Tables with Foreign Key Relationships in a Classic Replication Scheme

In a classic replication scheme, you may choose to replicate all or a subset of tables that have foreign key relationships with one another.

To do so, create the tables and the foreign key relationship on each master and subscriber. Then, add the tables to the replication scheme with the ALTER REPLICATION ADD ELEMENT statement on each master and subscriber.

However, if the foreign key relationships have been configured with ON DELETE CASCADE, then you must create all of the tables before the replication scheme is created. Then, configure the replication scheme with the CREATE REPLICATION statment to include all tables with either the DATASTORE element (that does not exclude any of the tables) or the TABLE element for every table that is involved in the relationship.

You cannot add a table with a foreign key relationship configured with ON DELETE CASCADE to the replication scheme after the replication scheme is created with the ALTER REPLICATION statement. Instead, you must drop the replication scheme, create the new table with the foreign key relationship with ON DELETE CASCADE, and then create a new replication scheme that includes all of the related tables.

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 Classic 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.

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 Classic 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 non-materialized view as a `DATASTORE` element, only the detail tables associated with the view are replicated. The view itself is not replicated. A matching view can be defined on the subscriber database, but is not required. If detail tables are replicated, TimesTen Classic 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 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 DATASTORE Element

A master database configured for asynchronous or return receipt replication is durable by default. This means that log records are committed to the file system 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 the file system 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 the file system.

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.

For example, you can replicate the entire contents of the master database (`masterds`) to the subscriber database (`subscriberds`) and eliminate the flush-log-to-disk operation by using `TRANSMIT NONDURABLE`. 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 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. Otherwise, you must follow the procedures described in Recovering Nondurable Databases to recover a failed nondurable database.

Using a Return Service in a Classic Replication Scheme

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.

See Using a Return Service.

Setting STORE Attributes in a Classic Replication Scheme

The `STORE` attributes clause in either the `CREATE REPLICATION` and `ALTER REPLICATION` statements are used to set optional behavior for return services, compression, timeouts, durable commit behavior, conflict reporting, and table definition checking.

For full details on how to use and configure the `STORE` attributes for a classic replication scheme, see Setting STORE Attributes. See CREATE ACTIVE STANDBY PAIR in the *Oracle TimesTen In-Memory Database SQL Reference* for a description of all `STORE` attributes.

Configuring Network Operations for a Classic Replication Scheme

If your replication host has more than one network interface, you may want to configure replication to use an interface other than the default interface.

For full details on how to configure more than one network interface for a classic replication scheme, see Configuring Network Interfaces with the ROUTE Clause.

Classic Replication Scheme Syntax Examples

The examples in this section illustrate how to configure a variety of classic replication schemes.

- Single Classic Subscriber Schemes
- Multiple Subscriber Classic Replication Schemes
- Replicating Tables to Different Subscribers
- Propagation Scheme
- Bidirectional Split Workload Schemes
- Bidirectional Distributed Workload Scheme

Single Classic Subscriber Schemes

The following example replicates a single table within a classic replication scheme. The example shows 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.

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

The following example replicates the entire databse. The scheme shown 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`.

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

Multiple Subscriber Classic Replication Schemes

You can create a classic replication scheme that includes up to 255 subscriber databases. If you are configuring propagator databases, you can configure up to 255 propagators. Each propagator can have up to 255 subscriber databases.

See Propagation Scheme for an example of a classic replication scheme with propagator databases.

This example replicates 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 classic 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";
```

This example replicates to two subscribers using the RETURN RECEIPT attribute and `STORE` parameters. 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;
```

This example enables RETURN RECEIPT for only one subscriber: `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;
```

This example shows how to enable different return services for subscribers. This example applies RETURN RECEIPT BY REQUEST to `subscriber1ds` and RETURN RECEIPT to `subscriber2ds`. In this classic replication 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 classic replication scheme establishes a master database, named `centralds`, which 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.

```
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

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 classic 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`.

```
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

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.

```
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

A bidirectional general workload classic 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.

```
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 Resolving Replication Conflicts.

The following example shows how to manage update conflicts. The `tstamp` timestamp column is included in the `ttuser.accounts` table. The `CREATE REPLICATION` statement has been modified to include the `CHECK CONFLICTS` clause.

```
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";
```

Applying a Classic Replication Scheme to a Database

When you define the classic replication scheme, save the `CREATE REPLICATION` statement in a SQL file. After you have described the classic replication scheme in a SQL file, you can run the SQL on the database using the `-f` option to the `ttIsql` utility.

The syntax is:

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

If your classic replication scheme is described in a file called `repscheme.sql`, you can run 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 classic replication scheme.

For example, if your classic replication 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 run the SQL file containing your classic replication scheme from the ttIsql command line after connecting to a database. For example:

```
run repscheme.sql;
```

Creating Classic Replication Schemes with Scripts

Creating your classic 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.

For example, consider a general workload bidirectional scheme. 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.

```
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 classic replication scheme with scripting. For example, the following perl script can be used to create the replication scheme shown in the previous example.

```
@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 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\."';
```

This Perl script example creates a classic 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.)

```
@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 Altering a Classic Replication Scheme

This chapter describes how to alter an existing classic replication system.

- Altering a Classic Replication Scheme
- Altering a Replicated Table in a Classic Replication Scheme
- Truncating a Replicated Table in a Classic Replication Scheme
- Dropping a Classic Replication Scheme

Table 10-1 lists the tasks often performed on an existing classic replicated system.

Table 10-1 Tasks Performed on an Existing Classic Replicated System

Task	What to Do
Alter or drop a classic replication scheme	See Altering a Classic Replication Scheme and Dropping a Classic Replication Scheme.
Alter a table used in a classic replication scheme	See Altering a Replicated Table in a Classic Replication Scheme.
Truncate a table used in a classic replication scheme	See Truncating a Replicated Table in a Classic Replication Scheme.
Change the replication state of a subscriber database	See Set the Replication State of Subscribers.
Resolve update conflicts	See Resolving Replication Conflicts.
Recover from failures	See Managing Database Failover and Recovery.
Upgrade database	Use the `ttMigrate` and `ttRepAdmin` utilities.

For more information on the `ttMigrate` and `ttRepAdmin` utilities, see Upgrades in TimesTen Classic in *Oracle TimesTen In-Memory Database Installation, Migration, and Upgrade Guide*.

Altering a Classic 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 to a Classic Replication Scheme.
- Add a PL/SQL object to the master database and implement its replication on subscribers. See Adding a PL/SQL Object to an Existing Classic Replication Scheme.

Use `ALTER REPLICATION` to alter the classic 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` built-in 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` built-in 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.

   For more information, see Starting and Stopping the Replication Agents.

If you use `ALTER REPLICATION` to change a classic 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 Classic Replication Scheme

There are two ways to add a table or sequence to an existing classic replication scheme.

- When the element level of the classic replication scheme is `TABLE` or `SEQUENCE`, use the `ALTER REPLICATION` statement with the `ADD ELEMENT` clause to add a table or sequence..
- When the element level of the classic replication scheme is `DATASTORE`, use the `ALTER REPLICATION` statement with the `ALTER ELEMENT` clause to include a table or sequence. .

This example alters the classic replication scheme `r1` to add sequence `seq` and table `westleads`, which are 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";
```

This example adds 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
    INCLUDE SEQUENCE my.seq
  ALTER ELEMENT ds1
    INCLUDE TABLE my.tab1;
```

Adding a PL/SQL Object to an Existing Classic Replication Scheme

You can add a new PL/SQL procedure, package, package body or function to an existing replication scheme.

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 Classic Replication Scheme

You can add a `DATASTORE` element to an existing classic replication scheme by using the `ALTER REPLICATION` statement with the `ADD ELEMENT` clause.

All tables except temporary tables, materialized views, and non-materialized 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 and Excluding a Table or Sequence When You Add a DATASTORE Element.

```
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 classic 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.

The following example adds the `ds1` `DATASTORE` element to `my.rep1` replication scheme. Then, it includes 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 classic 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.

The following example adds the `ds2` `DATASTORE` element to a replication scheme, but excludes 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 Classic Replication Scheme

You can drop a table or sequence that is replicated as part of a `DATASTORE`, `TABLE` or `SEQUENCE` element.

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

You can drop a table or sequence that is part of a classic replication scheme at the `DATASTORE` level.

1. Stop the replication agent.
2. Exclude the table or sequence from the `DATASTORE` element in the classic 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.

This example excludes the table `my.tab1` from the `ds1` `DATASTORE` element in the `my.rep1` replication scheme. Then drops the table.

```
ALTER REPLICATION my.rep1
  ALTER ELEMENT ds1
    EXCLUDE TABLE my.tab1;
DROP TABLE my.tab1;
```

Dropping a Table or Sequence That Is Replicated As a TABLE or SEQUENCE Element

You can drop a table that is part of a classic replication scheme at the `TABLE` or `SEQUENCE` level.

1. Stop the replication agent.
2. Drop the element from the classic replication scheme.
3. Drop the table or sequence.

This example drops the `SEQUENCE` element `elem_seq` from the classic replication scheme `r1`. Then, it drops the sequence `seq`.

```
ALTER REPLICATION r1
  DROP ELEMENT elem_seq;
DROP SEQUENCE seq;
```

Creating and Adding a Subscriber Database to a Classic Replication Scheme

You can add a new subscriber database while the replication agents are running.

To add a database to a classic 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. By default, any updates made to the master after the duplicate operation has started are also copied to the subscriber. Use the `-noSetMasterRepStart` option if you do not want to copy updates to the subscriber.
5. Start the replication agent on the newly created database (`ttAdmin -repStart`).

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 from a Classic Replication Scheme

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:

```
ALTER REPLICATION r1
  ALTER ELEMENT elem_westleads
    DROP SUBSCRIBER backup3 ON "backupserver";
```

Changing a TABLE or SEQUENCE Element Name in a Classic Replication Scheme

Stop the replication agent before you change a `TABLE` or `SEQUENCE` element name in a classic replication scheme.

Change the element name of the `westleads` table from `elem_westleads` to `newelname`:

```
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 in a Classic Replication Scheme

Stop the replication agent before you replace a master database in a classic replication scheme.

In this example, `newwestds` is made the new master for all elements currently configured for the master, `westds`:

```
ALTER REPLICATION r1
  ALTER ELEMENT * IN westds
    SET MASTER newwestds;
```

Eliminating Conflict Detection in a Classic Replication Scheme

In this example, conflict detection configured by the `CHECK CONFLICTS` clause in the classic replication scheme shown is eliminated for the `elem_accounts_1` table.

```
ALTER REPLICATION r1
  ALTER ELEMENT elem_accounts_1
    SET NO CHECK;
```

See Resolving Replication Conflicts.

Eliminating the Return Receipt Service in a Classic Replication Scheme

In this example, the return receipt service is eliminated for the first subscriber in the classic replication scheme.

```
ALTER REPLICATION r1
  ALTER ELEMENT elem_waccounts
    ALTER SUBSCRIBER eastds ON "eastcoast"
      SET NO RETURN;
```

Changing the Port Number for a Classic Replication Scheme

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 for details on how to assign ports to the replication agents.

In this example, the `r1` replication scheme is altered to change the port number of the `eastds` to 22251:

```
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. See Configuring Network Interfaces with the ROUTE Clause.

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` to define a new threshold value, and then restart the replication agents.

See Setting the Transaction Log Failure Threshold.

Altering a Replicated Table in a Classic Replication Scheme

You can use `ALTER TABLE ... ADD COLUMN` or `ALTER TABLE ... DROP COLUMN` statements to add or drop columns on the master database in a classic replication scheme.

The `ALTER TABLE` operation is replicated to alter the subscriber databases. These are the only `ALTER TABLE` clauses that are replicated in a classic replication scheme.

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` built-in procedure or monitoring tools described in Managing Replication to confirm the updates have been received and committed on the databases.

Also, if you are running 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 in a Classic Replication Scheme

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 Classic Replication Scheme

You can use the `DROP REPLICATION` statement to remove a replication scheme from a database. You cannot drop a classic replication scheme when master catchup is required unless it is the only classic 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 classic 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 classic replication scheme *before* dropping the replicated tables. Otherwise, you receive an error indicating that you have attempted to drop a replicated table or index.

To remove the `tab` table and `repscheme` replication scheme from a database, enter the following:

```
DROP REPLICATION repscheme;
DROP TABLE tab;
```

11

Managing Replication

This chapter describes some of the TimesTen utilities and built-in procedures you can use to monitor and manage the replication status of your databases.

You can manage and 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 manage and monitor replication from your programs, you can use the TimesTen built-in procedures described in Built-In Procedures in *Oracle TimesTen In-Memory Database Reference* or create your own SQL `SELECT` statements to query the replication tables described in Replication Tables 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.

- Using ttStatus to Obtain Replication Agent Status
- Using ttAdmin -query to Confirm Policy Settings
- Using ttDataStoreStatus to Obtain Replication Agent Status

You can also obtain the state of specific replicated databases as described in Show Subscriber Database Information and Show the Configuration of Replicated Databases.

Using ttStatus to Obtain Replication Agent Status

Use the `ttStatus` utility to confirm that the replication agent is started for the master database.

```
> 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.

```
> 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
```

The replication restart policy is described in Starting and Stopping the Replication Agents.

Using ttDataStoreStatus to Obtain Replication Agent Status

Call the `ttDataStoreStatus` built-in procedure 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.

Replication of Statistics

You often compute statistics before preparing your statements, since the information is likely to result in a more efficient query optimizer plan.

Both SQL statistics and performance statistics are not replicated to any secondary databases. Instead, you must compute statistics on each database independently.

See Compute Exact or Estimated Statistics in the *Oracle TimesTen In-Memory Database Operations Guide*.

Set 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 Show State of Replication Agents.

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 11-1.

Table 11-1 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. **WARNING:** If you are planning on restarting this subscriber, updates are not stored between the stop and the restart. Therefore, when you restart, the subscriber does not contain all of the updates from the master. If you are planning to restart, pause the subscriber instead of stopping it.

Table 11-1 (Cont.) Database States

State	Description
`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.

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 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.

To use `ttRepAdmin` from the command line to set the subscriber state. This example directs 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.

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`:

```
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.

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
```

This example uses `ttRepAdmin` to display information about the master database described in Multiple Subscriber Classic Replication Schemes.

```
> 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 a `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 Classic Replication Schemes. 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 transaction log data accumulates.

Part of the output from the queries described in this section enables you to see how much transaction log data has accumulated on behalf of each subscriber database and the amount of time since the last successful communication with each subscriber database.

Use the following methods to display information for subscriber databases:

- Display Subscriber Status with the ttRepAdmin Utility
- Display Subscriber Status with the ttReplicationStatus Built-In Procedure
- Display Information About Subscribers Through Querying Replication Tables

Display Subscriber Status with the ttRepAdmin Utility

To display information about subscribers, use the `ttRepAdmin` utility with the `-receiver -list` options.

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

This example uses the `ttRepAdmin` utility to display information about the subscribers described in Multiple Subscriber Classic Replication Schemes.

```
> 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.

Note

See Subscriber Information.

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

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

See ttRepAdmin in the *Oracle TimesTen In-Memory Database Reference.*

Display Subscriber Status with the ttReplicationStatus Built-In Procedure

Within ttIsql, you can display status for a one or more subscriber databases by using the `ttReplicationStatus` built-in procedure, which reports only on the status of the subscribers for the master database on which this built-in procedure is called.

The following retrieves status for the subscriber `master2` that is located on `host1`. If the host name is excluded, the subscriber is located solely on its name.

```
Command> call ttReplicationStatus('master2', 'host1');
< MASTER2, HOST1, 0, start, 1, 26, _ACTIVESTANDBY , TTREP >
1 row found.
```

The information shown is that the subscriber `master2` located on `host1` that is listening on an automatically assigned port. The TCP/IP port is used by the subscriber agent to receive updates from the master. However, since the value is zero, this indicates replication has automatically assigned the port.

This subscriber is in the start state. There is only one transaction log being held for this peer and 26 seconds have passed since the last replication. The name of the replication scheme is `_ACTIVESTANDBY` and the owner is `TTREP`.

If you do not provide either a subscriber or the subscriber host names, then the status for all subscribers of this master are returned. The following shows the same status as above, since there is only one subscriber set up for this master.

```
Command> call ttReplicationStatus();
< MASTER2, HOST1, 0, start, 1, 26, _ACTIVESTANDBY , TTREP >
1 row found.
```

See ttReplicationStatus in the *Oracle TimesTen In-Memory Database Reference.*

Display Information About Subscribers Through Querying Replication Tables

You can obtain the same information about a master's subscribers from a program by querying the `TTREP.REPPEERS`, `TTREP.TTSTORES`, and `SYS.MONITOR` tables with a `SELECT` statement.

```
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 >
```

See Subscriber Information.

Subscriber Information

The output from either the `ttRepAdmin` utility or the `SELECT` statement contains fields describing the subscriber database.

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.

Field	Description
`State`	Current replication state of the subscriber with respect to its master database (see Show Subscriber Database 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.
`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 with certain tools.

- Display Configuration Information with the ttIsql repschemes Command
- Display Configuration Information with the ttRepAdmin Utility
- Display Configuration Information Through Querying Replication Tables

Display Configuration Information with the ttIsql repschemes Command

To display the configuration of your replicated databases from the `ttIsql` prompt, use the `repschemes` command.

```
repschemes;
```

The following example shows the configuration output from the replication scheme shown in Propagation Scheme.

```
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
```

Display Configuration Information with the ttRepAdmin Utility

To display the configuration of your replicated databases, use the `ttRepAdmin` utility with the `-showconfig` option.

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

The following shows the configuration output from the propagated databases configured by the replication scheme shown in Propagation Scheme. The `propds` propagator shows a latency of 19.41 seconds and is 2 logs behind the master.

```
> 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 Display Information About Subscribers Through Querying Replication Tables 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.

Display Configuration Information Through Querying Replication Tables

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;
```

The output from the queries refer to the databases configured by the replication scheme shown in Propagation Scheme.

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 Transaction Log Failure Threshold.

The output from the second query might be:

```
< propds, nethandler, 0, 0, 7, 1004378953, 0, -1.00000000000000, -1, -1, 1 >
```

See Display Information About Subscribers Through Querying Replication Tables 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

You can monitor replication through bookmarks and the log sequence numbers with certain tools.

In a replicated database, transactions remain in the transaction log buffer and transaction log files until the master replication agent confirms they have been fully processed by the subscriber. In an active standby pair replication scheme that contains subscribers, transactions remain in the transaction logs until the active master confirms that they are processed by both the standby master and any subscribers. Only then can the active 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.

Note

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.

You can monitor replication through bookmarks and the log sequence numbers with the following tools:

- Monitor Replication with the TTREP.REPPEERS Table
- Monitor Replication from the Replication Log Holds
- Monitor Replication with the ttRepAdmin Utility
- Monitor Replication with the ttBookMark Built-In Procedure

Monitor Replication with the TTREP.REPPEERS Table

An 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.

Monitor Replication from the Replication Log Holds

Select from the `SYS.V$LOG_HOLDS` system view or call the `ttLogHolds` built-in procedure to get information about replication log holds.

The following example shows the output of `ttLogHolds` built-in procedure for an active standby pair replication scheme, where the active master is `master1` and the standby master is `master2`, with a single subscriber, `subscriber1`. All transactions are replicated from the active master first to the standby master who then propagates the transactions to the subscriber. Thus, the subscriber's progress is slightly behind the standby master's progress.

The active master monitors the progress of both the standby master and the subscriber; therefore, if the standby master goes down for any reason, the active master can take over the replication to the subscriber. The active master receives acknowledgements when transactions are applied on the subscriber so the active master knows when it can release pertinent log records that might be needed if the standby master fails (upon which the active master switches to replicate directly to the subscribers). The transactions remain in the transaction logs until they are processed on both the standby master and the subscriber.

```
Command> call ttLogHolds;
< 0, 3569664, Checkpoint                    , master1.ds0 >
< 0, 15742976, Checkpoint                    , master1.ds1 >
< 0, 16351496, Replication                   , ADC6160529:SUBSCRIBER1 >
< 0, 16351640, Replication                   , ADC6160529:MASTER2 >
4 rows found.
```

If you are using an AWT cache group, it uses the replication agent to asynchronously propagate transactions to the Oracle database. When you call the `ttLogHolds` built-in procedure, the description field contains "`_ORACLE`" to identify the transaction log hold for the AWT cache group propagation.

```
Command> call ttLogHolds();
< 0, 18958336, Checkpoint                    , cachealone1.ds0 >
< 0, 19048448, Checkpoint                    , cachealone1.ds1 >
< 0, 19050904, Replication                   , ADC6160529:_ORACLE >
3 rows found.
```

See SYS.V$LOG_HOLDS in the *Oracle TimesTen In-Memory Database System Tables and Views Reference* or ttLogHolds in the *Oracle TimesTen In-Memory Database Reference*.

Monitor Replication with the ttRepAdmin Utility

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). If you are monitoring an active standby pair with one or more subscribers, then this value denotes the oldest record held for all nodes involved in the replication scheme. For example, in an active standby pair with subscribers, the oldest record could be held in the log for the standby master or any of the subscribers.
`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.

Monitor Replication with the ttBookMark Built-In Procedure

Use the `ttBookmark` built-in procedure to display the location of bookmarks.

```
> 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."

If you are monitoring an active standby pair with one or more subscribers, then the "Replication hold LSN" denotes the oldest record held for all nodes involved in the replication scheme. For example, in an active standby pair with subscribers, the oldest record could be held in the log for the standby master or any of the subscribers.

Use ttRepAdmin to Show Replication Status

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.

The following example demonstrates the `ttRepAdmin -showstatus` output.

Following the examples 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

The following example is an active standby pair replication scheme in which the `rep1` database is the active master and `rep2` database is the standby master in an active standby pair replication scheme.

The `ReplicationParallelism` connection attribute in this example is set to 2, which indicates the number of transmitter threads on the source database and the number of receiver threads on the target database.

```
$ ttRepAdmin -showstatus -detail rep1

Replication Agent Status as of: 2019-09-03 10:01:31

DSN                         : rep1
Process ID                  : 9012 (Started)
Replication Agent Policy    : manual
Host                        : MYHOST
RepListener Port            : 56209 (AUTO)
Main thread's state         : STATE_MM_IDLE_STATE
Last write LSN              : 0.22169832
Last LSN forced to disk     : 0.22169600
Replication hold LSN        : 0.22159592
```

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.

```
Replication Peers:
   Name                     : REP2
   Host                     : MYHOST2
   Port                     : 51509 (AUTO) (Connected)
   Replication State        : STARTED
   Communication Protocol   : 47

   Name                     : REP2
   Host                     : MYHOST2
   Port                     : 51509 (AUTO) (Connected)
   Replication State        : STARTED
   Communication Protocol   : 47
```

Since `ReplicationParallelism` is set to 2, there are two entries for the replication peer for `rep2`.

```
REPHOLD thread (REPHOLD:140175402608384):
   Start/Restart count   : 1
   Current state         : STATE_REPHOLD_SLEEPING
   Current DB context    : 0x7f7d180008c0

REPLISTENER thread (REPLISTENER:140175393158912):
   Start/Restart count   : 1
   Current state         : STATE_LISTENER_WAIT_FOR_PEER_CONN
   Current DB context    : 0x7f7c9c0008c0
   Most recent errors (max 5):
     TT16999 in receiver.c (line 2608) at 09:55:38 on 09-03-2019
     TT16999 in receiver.c (line 2608) at 09:55:55 on 09-03-2019

LOGFORCE thread (LOGFORCE:140175407265536):
   Start/Restart count   : 1
   Current state         : STATE_LOGFORCE_SLEEPING
   Current DB context    : 0x7f7d140afae0
```

With an active standby pair, each master has both transmitters to and receivers from the other master. The number of transmitters and receivers determined by the `ReplicationParallelism` setting. In this example, since `ReplicationParallelism` is set to 2, there are two transmitter and receiver threads between the active and standby masters, shown as track 0 and track 1 in the output.

```
TRANSMITTER thread(s) (TRANSMITTER(S):140172775343872):
 For                     : REP2 (track 1)(SSL)
 Start/Restart count   : 1
   Current state         : STATE_XMTR_FLUSH_SENDING_DONE
   Current DB context    : 0x7f7ca44ff640
   Send LSN              : 0.22159592
   Replication Tables CTN: 1567529721.542
   Transactions sent     : 101
   Total packets sent    : 36
   Tick packets sent     : 25
   MIN sent packet size  : 64
   MAX sent packet size  : 16384
   AVG sent packet size  : 876
   Last packet sent at   : 10:01:26
   Total Packets received: 34
   MIN rcvd packet size  : 64
   MAX rcvd packet size  : 128
   AVG rcvd packet size  : 119
   Last packet rcvd'd at : 10:01:26
   TXNs Allocated        : 102
   TXNs In Use           : 0
   ACTs Allocated        : 101
   ACTs In Use           : 0
   ACTs Data Allocated   : 0
   Timeout               : 7200
   Adapted Timeout Max   : 7200
   Adapted Timeout Time  : 1567530021
   current txn           : 0.0
   Longest batch runtime : 0
   Longest batch 1st txn : 0.0
   Longest batch lst txn : 0.0
   Largest txn (ops)     : 1567529721.441
   Largest txn (#ops)    : 1
   Longest txn (time)    : 0.0
   Longest txn (secs)    : 0
```

```
      Most recent errors (max 5):
        TT16999 in transmitter.c (line 1465) at 09:55:56 on 09-03-2019

   RECEIVER thread(s) (RECEIVER:140173241992960):
    For                   : REP2 (track 1)(SSL)
      Start/Restart count   : 1
      Current state         : STATE_RCVR_READ_NETWORK_LOOP
      Current DB context    : 0x7f7c9c4b0b90
      Transactions received : 0
      Total packets sent    : 28
      Tick packets sent     : 0
      MIN sent packet size  : 64
      MAX sent packet size  : 128
      AVG sent packet size  : 122
      Last packet sent at   : 10:01:19
      Total Packets received: 29
      MIN rcvd packet size  : 64
      MAX rcvd packet size  : 156
      AVG rcvd packet size  : 68
      Last packet rcvd'd at : 10:01:19
      rxWaitCTN             : 0.0
      prevCTN               : 0.0
      current txn           : 0.0
      STA Blk Data Allocated: 0
      STA Data Allocated    : 0
      Longest batch runtime : 0
      Longest batch 1st txn : 0.0
      Longest batch lst txn : 0.0
      Largest txn (ops)     : 0.0
      Largest txn (#ops)    : 0
      Longest txn (time)    : 0.0
      Longest txn (secs)    : 0

   TRANSMITTER thread(s) (TRANSMITTER(M):140175390009088):
    For                   : REP2 (track 0)(SSL)
      Start/Restart count   : 1
      Current state         : STATE_META_PEER_INFO
      Current DB context    : 0x7f7c980008c0
      Send LSN              : 0.22159592
      Replication Tables CTN: 1567529721.386
      Transactions sent     : 100
      Total packets sent    : 88
      Tick packets sent     : 74
      MIN sent packet size  : 64
      MAX sent packet size  : 16384
      AVG sent packet size  : 394
      Last packet sent at   : 10:01:26
      Total Packets received: 86
      MIN rcvd packet size  : 64
      MAX rcvd packet size  : 128
      AVG rcvd packet size  : 123
      Last packet rcvd'd at : 10:01:26
      TXNs Allocated        : 101
      TXNs In Use           : 0
      ACTs Allocated        : 100
      ACTs In Use           : 0
      ACTs Data Allocated   : 0
      Timeout               : 7200
      Adapted Timeout Max   : 7200
      Adapted Timeout Time  : 1567529956
      current txn           : 0.0
      Longest batch runtime : 0
      Longest batch 1st txn : 0.0
      Longest batch lst txn : 0.0
      Largest txn (ops)     : 1567529721.286
      Largest txn (#ops)    : 1
      Longest txn (time)    : 0.0
      Longest txn (secs)    : 0
      Most recent errors (max 5):
        TT16999 in transmitter.c (line 1465) at 09:55:38 on 09-03-2019

   RECEIVER thread(s) (RECEIVER:140173245142784):
    For                     : REP2 (track 0)(SSL)
      Start/Restart count   : 1
      Current state         : STATE_RCVR_READ_NETWORK_LOOP
      Current DB context    : 0x7f7c944a41f0
      Transactions received : 0
      Total packets sent    : 79
      Tick packets sent     : 0
      MIN sent packet size  : 64
      MAX sent packet size  : 128
      AVG sent packet size  : 125
      Last packet sent at   : 10:01:29
      Total Packets received: 80
      MIN rcvd packet size  : 64
      MAX rcvd packet size  : 156
      AVG rcvd packet size  : 65
      Last packet rcvd'd at : 10:01:29
      rxWaitCTN             : 0.0
      prevCTN               : 0.0
      current txn           : 0.0
      STA Blk Data Allocated: 0
      STA Data Allocated    : 0
      Longest batch runtime : 0
      Longest batch 1st txn : 0.0
      Longest batch lst txn : 0.0
      Largest txn (ops)     : 0.0
      Largest txn (#ops)    : 0
      Longest txn (time)    : 0.0
      Longest txn (secs)    : 0
```

MAIN Thread Status Fields

The `MAIN` thread status 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
`Host`	Name of the machine that hosts this database.
`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.
`Main thread's state`	Internal use only.

MAIN Thread	Description
`Last write LSN`	The location of the most recently generated transaction log record for the database. See Show Replicated Log Records.
`Last LSN forced to disk`	The location of the most recent transaction log record written to the disk. See Show Replicated Log Records.
`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.

Replication Peer Status Fields

The replication peer status 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.
`Communication Protocol`	Internal protocol used by replication to communicate between the peers. (For internal use only.)

Note

The `REPHOLD`, `REPLISTENER` and `LOGFORCE` thread sections are for internal use only.

TRANSMITTER Thread Status Fields

The `TRANSMITTER` thread status 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. Also, whether the replication agent transmitters are using Transport Layer Security (TLS) (indicated with "SSL").
`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.
`Current state`	Internal use only.
`Current DB context`	Internal use only.
`Send LSN`	The last LSN transmitted to this peer. See Show Replicated Log Records.
`Replication Tables CTN`	Internal use only.
`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).

`TXNs Allocated`
`TXNs In Use`
`ACTs Allocated`
`ACTs In Use`
`ACTs Data Allocated`
`Timeout`
`Adapted Timeout Max`
`Adapted Timeout Time`
`current txn`
`Longest batch runtime`
`Longest batch 1st txn`
`Longest batch lst txn`
`Largest txn (#ops)`
`Longest txn (time)`
	Internal use only.

`Largest txn (ops)`	Number of operations.
`Longest txn (secs)`	Amount of time in seconds taken to process the longest transaction.
`Most recent errors (max 5)`	Last five errors generated by this thread.

RECEIVER Thread Status Fields

The `RECEIVER` thread status 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. Also, whether the replication agent receivers are using Transport Layer Security (TLS) (indicated with "SSL").
`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.
`Current state`	Internal use only.
`Current DB context`	Internal use only.
`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).

RECEIVER Thread	Description
`rxWaitCTN`	Internal use only.
`prevCTN`	
`current txn`	
`STA Blk Data Allocated`	
`STA Data Allocated`	
`Longest batch runtime`	
`Longest batch 1st txn`	
`Longest batch lst txn`	
`Largest txn (#ops)`	
`Longest txn (time)`	
`Largest txn (ops)`	Number of operations.
`Longest txn (secs)`	Amount of time in seconds taken to process the longest transaction.

Check the Status of Return Service Transactions

You can determine the status of a return service or to find out what the last returned status was.

- Determine If Return Service Is Disabled
- Check Last Returned Status for a Return Service

Determine If Return Service Is Disabled

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.

The `ttRepSyncSubscriberStatus` built-in 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.

This example uses `ttRepSyncSubscriberStatus` to obtain the return receipt status of the `subscriberds` database with respect to its master database, `masterDSN`.

```
> ttIsql masterDSN

Command> CALL ttRepSyncSubscriberStatus ('subscriberds');
< 0 >
1 row found.
```

This result indicates that the return service is still enabled.

See Disabling Return Service Blocking Manually.

Check Last Returned Status for a Return Service

You can check the status of the last return receipt or return twosafe transaction processed on the connection handle by calling the `ttRepXactTokenGet` and `ttRepXactStatus` built-in procedures.

First, call the `ttRepXactTokenGet` built-in procedure 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` built-in procedure to obtain the return service status. The output of the `ttRepXactStatus` built-in 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` built-in 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. This is described in Using a Return Service.

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` built-in procedure returns the return service status for each subscriber as a set of rows formatted as:

```
subscriberName, status, error
```

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 processed 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);

    /* run */
    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;
}
```

Analyze Outstanding Transactions in the Replication Log

You can use the `-logAnalyze` command in the `ttXactLog` utility to analyze the replication logs.

You can determine the following:

- Measure how much is left to replicate from a master to any subscribers at the current time. When replication seems to be taking longer than expected, you can determine how many transactions are left to replicate or if replication is processing a long-running transaction.
- Measure if the current configuration distributes the load appropriately across all manual and automatic tracks for parallel replication.

Run the log analyze command against a particular data store to generate the following information:

- The number of transactions that are waiting to be replicated. For each transaction that has not been replicated, the information collected includes the number of operations for each transaction and the total size of each transaction (including partial roll backs).
- The amount of operations left in each transaction including its specific type (either DDL or DML) and how many of each statement type are in each transaction. The tool also generates the total size of every operation left to replicate.

- Retrieves information on how the workload is split across tracks. When you use manual parallel replication, you can use `-logAnalyze` to monitor whether the application is distributing work evenly across the replication tracks.
- Returns the largest transaction left to replicate.
- Returns the start and end LSN. The start LSN is the starting point in the transaction log where the transmitter starts reading; the end LSN is the end of the transaction log.

Note

In the transaction log analysis output, transactions are shown in commit order.

You can specify how much information is displayed with the verbose command. For example, when you set verbose to 1, the following is displayed:

```
% ttXactLog -v1 -logAnalyze rep1
Summary:
Total transactions left to replicate: 4
Total rows left to replicate: 4
Size of transactions left to replicate: 1.86 KiB
Size of rows left to replicate: 488.00 B
Total inserts remaining: 4
Total partial rollbacks: 5
Total rollbacks: 3

Start LSN = 0.3793736
End LSN = 0.18769920
```

If a subscriber is specified, then the summary displays only for that particular subscriber. However, by default, the summary is displayed for all subscribers.

When you specify verbose to 2, then the information includes both a summary of each parallel track in addition to the overall summary information.

```
% ttXactLog -v2 -logAnalyze rep1

Track analysis for track number: 0
Transactions left to replicate: 2
Rows left to replicate: 2
Size of transactions left to replicate: 880.00 B
Size of rows left to replicate: 312.00 B
Total inserts remaining: 1
Total partial rollbacks: 4

Track analysis for track number: 1
Transactions left to replicate: 2
Rows left to replicate: 2
Size of transactions left to replicate: 1.14 KiB
Size of rows left to replicate: 244.00 B
Total inserts remaining: 2
Total partial rollbacks: 1
Total rollbacks: 3

Summary:
Total transactions left to replicate: 4
Total rows left to replicate: 4
Size of transactions left to replicate: 2.00 KiB
Size of rows left to replicate: 556.00 B
```

```
Total inserts remaining: 3
Total partial rollbacks: 5
Total rollbacks: 3

Start LSN = 0.3793736
End LSN = 0.20506624
```

When you provide the verbose level 3, the information generated includes a transaction analysis that includes a description of the contents of every transaction in every track:

```
% ttXactLog -v3 -logAnalyze rep1

Transaction id:    3.10
Track for this xid: 1
Logmarker before this xid: 275
Rows left to replicate: 1
Transaction size: 800.00 B
Size of rows left: 122.00 B
Total inserts remaining: 1

Transaction id:    2.1
Track for this xid: 0
Logmarker before this xid: 276
Rows left to replicate: 1
Transaction size: 368.00 B
Size of rows left: 122.00 B
Total inserts remaining: 1

Transaction id:    2.19
Track for this xid: 1
Logmarker before this xid: 823
Rows left to replicate: 1
Transaction size: 368.00 B
Size of rows left: 122.00 B
Total inserts remaining: 1

Transaction id:    3.2
Track for this xid: 0
Logmarker before this xid: 842
Rows left to replicate: 1
Transaction size: 368.00 B
Size of rows left: 122.00 B
Total inserts remaining: 1

Track analysis for track number: 0
Transactions left to replicate: 2
Rows left to replicate: 2
Size of transactions left to replicate: 736.00 B
Size of rows left to replicate: 244.00 B
Total inserts remaining: 2
Total partial rollbacks: 4

Track analysis for track number: 1
Transactions left to replicate: 2
Rows left to replicate: 2
Size of transactions left to replicate: 1.14 KiB
Size of rows left to replicate: 244.00 B
Total inserts remaining: 2
Total partial rollbacks: 1
Total rollbacks: 3

Summary:
```

```
Total transactions left to replicate: 4
Total rows left to replicate: 4
Size of transactions left to replicate: 1.86 KiB
Size of rows left to replicate: 488.00 B
Total inserts remaining: 4
Total partial rollbacks: 5
Total rollbacks: 3

Start LSN = 0.3793736
End LSN = 0.21444608
```

When you specify the XID, the tool displays verbose level 3 output where the transaction analysis is based on the XID. If an XID is used by two separate transactions, the report shows with the `LogMarker` entry the nearest point in the transaction log just before the start of each transaction.

```
% ttXactLog -logAnalyze -xid 2.19 rep1;

Transaction id:    2.19
Track for this xid: 1
Logmarker before this xid: 823
Rows left to replicate: 1
Transaction size: 368.00 B
Size of rows left: 122.00 B
Total inserts remaining: 1

Track analysis for track number: 0
Transactions left to replicate: 0
Rows left to replicate: 0
Size of transactions left to replicate: 0.00 B
Size of rows left to replicate: 0.00 B

Track analysis for track number: 1
Transactions left to replicate: 1
Rows left to replicate: 1
Size of transactions left to replicate: 368.00 B
Size of rows left to replicate: 122.00 B
Total inserts remaining: 1

Summary:
Total transactions left to replicate: 1
Total rows left to replicate: 1
Size of transactions left to replicate: 368.00 B
Size of rows left to replicate: 122.00 B
Total inserts remaining: 1

Start LSN = 0.3793736
End LSN = 0.20514816
```

Note

See ttXactLog in the *Oracle TimesTen In-Memory Database Reference*.

12 Resolving Replication Conflicts

In order to resolve replication conflicts, you need to understand how they occur and how they are reported.

This chapter includes these topics:

- How Replication Conflicts Occur
- Using a Timestamp to Resolve Conflicts
- Configuring Timestamp Comparison
- Reporting Conflicts
- The Conflict Report XML Document Type Definition

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 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

Update and insert conflicts can occur under several circumstances.

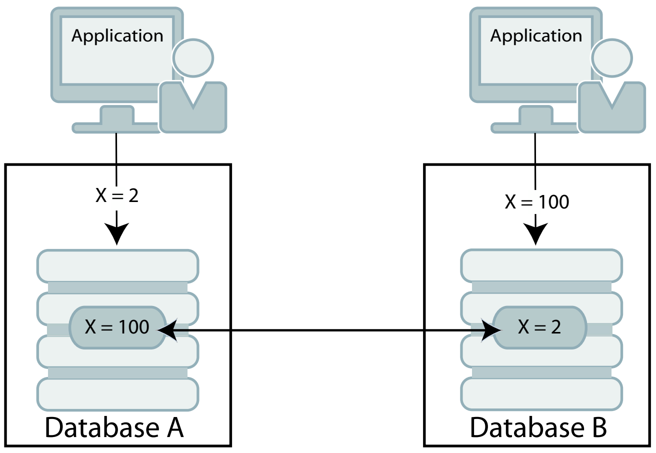
Figure 12-1 shows the results from an update conflict, which would occur for the value of X under the following circumstances:

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 12-1 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.

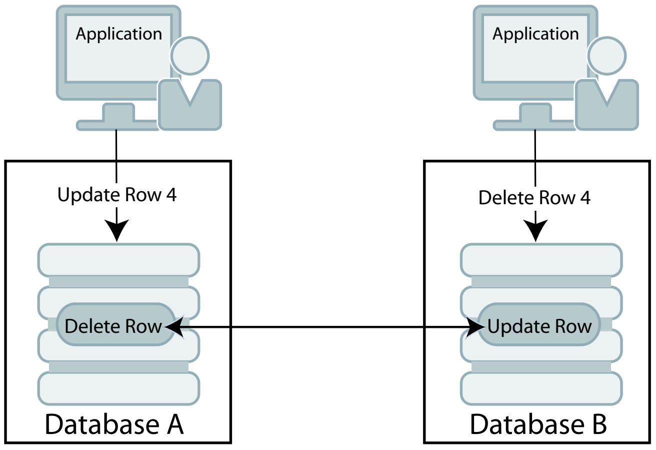
TimesTen Classic replication uses timestamp-based conflict resolution to cope with simultaneous updates or inserts. Through the use of timestamp-based conflict resolution, you may be able to keep the replicated databases synchronized and transactionally consistent.

Delete/Update Conflicts

Figure 12-2 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 12-2 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.

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.

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.

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 or Linux 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.

TimesTen uses the time value returned by the system at the time the transaction applies each record 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 a transaction received from a master, the replication agent at the subscriber database performs timestamp resolution for insert, update, and delete operations as follows:

- When applying replicated `INSERT` operations:
  - If the timestamp of the transaction record to be applied is newer than the timestamp of the existing row, the existing row is overwritten.
  - If the timestamp of the transaction record and the stored record are equal, the insert operation is discarded.
  - If the timestamp of the transaction record is older than the timestamp of the stored record, the insert operation from the transaction is discarded.
  - If the stored row has been deleted, then no timestamp is available for comparison. A replicated insert operation on a previously deleted row is applied as an insert.
- When applying replicated `UPDATE` operations:
  - If the timestamp of the transaction record to be applied is newer than the timestamp of the stored record, TimesTen applies the update operation to the row.
  - If the timestamp of the transaction record and the stored record are equal, the update operation is discarded.

  - If the timestamp of the transaction record is older than the timestamp of the stored record, the update operation from the transaction is discarded.
  - If the stored row has been deleted, then no timestamp is available for comparison. Any replicated update operations designated for the deleted row are discarded.
  - An update operation that cannot find the updated row is considered a delete conflict, which is reported but cannot be resolved.
- When applying replicated `DELETE` operations:
  - If the timestamp of a replicated delete operation is newer than the timestamp of the existing row, the existing row is deleted.
  - If the timestamp of a replicated delete operation is older than the timestamp of the stored record (the row has been modified recently), the delete operation is rejected.

Note

If the `ON EXCEPTION NO ACTION` clause is specified for a table, then the update, insert, or delete operation that fails a timestamp comparison is rejected. This may result in transactional inconsistencies if replication applies some (but not all) of the actions of a transaction. If the `ON EXCEPTION ROLLBACK WORK` clause is specified for a table (which is also the default), an update operation that fails timestamp comparison causes the entire transaction to be rejected.

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

You can set up your replication scheme for 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.
- 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.

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. This example shows that the `rep.tab` table contains a column named `tstamp` of type `BINARY(8)` to hold the timestamp value.

```
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.

For more details on the syntax, see CREATE REPLICATION statement and CHECK CONFLICTS clause in the *Oracle TimesTen In-Memory Database SQL Reference*. The following example shows how `CHECK CONFLICTS` might be used when configuring your replication scheme.

In this example, we establish automatic timestamp comparison for the bidirectional replication scheme. 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 in a Classic Replication Scheme and apply to each replicated database.

Enabling System Timestamp Column Maintenance

You can enable system timestamp comparison.

```
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

You can enable user timestamp column maintenance on a table.

```
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

You can configure replication to report conflicts to a human-readable text file (the default).

```
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, Reporting Update Conflicts and Reporting Delete/Update Conflicts.

Reporting Conflicts to an XML File

You can configure replication to report conflicts to an XML file.

```
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.

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
```

This example 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.

```
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
```

The following example 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.

```
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
```

The following example 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.

```
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
```

The following example 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.

```
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

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.

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 write operations to the conflict report file.

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. If the replication agent is active and you set `CONFLICT REPORTING RESUME AT` to `0`, reporting does not resume until the replication agent is restarted.

The following example 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.

```
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 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.

The TimesTen XML format conflict report is are based on the XML 1.0 specification (http://www.w3.org/TR/REC-xml/).

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.

The following example illustrates the format of a uniqueness conflict XML 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">
              <columnname>TSTAMP</columnname>
              <columntype>BINARY(8)</columntype>
              <columnvalue>3C9F983D00031128</columnvalue>
           </column>
        </conflictingtuple>
        <keyinfo>
           <column pos="1">
             <columnname>COL1</columnname>
             <columntype>NUMBER(38)</columntype>
             <columnvalue>2</columnvalue>
           </column>
       </keyinfo>
    </conflict>
    <scope>TRANSACTION</scope>
    <failedtransaction>
      <insert>
         <sql>Insert into table 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.

The following example illustrates the format of an update conflict XML 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</columnname>
          <columntype>NUMBER(38)</columntype>
          <columnvalue>99</columnvalue>
        </column>
        <column pos="3">
          <columnname>TSTAMP</columnname>
          <columntype>BINARY(8)</columntype>
          <columnvalue>3C9FACB600085CA0></columnvalue>
        </column>
     </existingtuple>
     <conflictingtuple>
        <column pos="3">
          <columnname>TSTAMP</columnname>
          <columntype>BINARY(8)</columntype>
          <columnvalue>3C9FACB6000612B0</columnvalue>
        </column>
        <column pos="2">
          <columnname>COL2</columnname>
          <columntype>NUMBER(38)</columntype>
          <columnvalue>50</columnvalue>
        </column>
    </conflictingtuple>
    <oldtuple>
        <column pos="3">
          <columnname>TSTAMP</columnname>
          <columntype>BINARY(8)</columntype>
          <columnvalue>3C9FAC85000E01F0</columnvalue>
       </column>
       <column pos="2">
          <columnname>COL2</columnname>
          <columntype>NUMBER(38)</columntype>
          <columnvalue>2</columnvalue>
       </column>
   </oldtuple>
   <keyinfo>
       <column pos="1">
         <columnname>COL1</columnname>
         <columntype>NUMBER(38)</columntype>
         <columnvalue>6</columnvalue>
       </column>
  </keyinfo>
</conflict>
```

```
<scope>TRANSACTION</scope>
<failedtransaction>
   <update>
      <<sql>Update table 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 for a description of the information that is written to the conflict report for a delete/update conflict.

The following example illustrates the format of a delete or update conflict XML element in which an update fails because the row has been deleted more recently.

```
<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>
```

The following example illustrates the format of a delete/update conflict XML element in which a deletion fails because the row has been updated more recently.

```
<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>
```

13

Improving Replication Performance

There are methods you can implement to improve replication performance.

The following describes methods for increasing replication performance:

- Configure parallel replication. See Configuring Parallel Replication.
- Use asynchronous replication, which is the default. See Making Decisions About Performance and Recovery Tradeoffs. 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. See Setting Connection Attributes for Logging.
- If the workload is heavy enough that replication sometimes falls behind, replicated changes must be captured from the transaction logs 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*.
- Adjust the transaction log buffer size and CPU power and resources. See Adjust Transaction Log Buffer Size and CPU.
- There can be performance issues after altering tables with multiple partitions and extraneous space. See Performance Considerations When Altering Tables That Are Replicated.
- Increase the number of threads that apply changes from the active master database to the standby master database by altering the `RecoveryThreads` first connection attribute. See Increase Replication Throughput for Active Standby Pairs.
- Replication and XLA operations have significant overhead with transaction logging. Replication scales best when there are a limited number of transmitters or receivers. See Limit Replication Transmitters, Receivers, and XLA Readers.

Note

Additional recommendations can be found in Poor Replication or XLA Performance in *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

Adjust Transaction Log Buffer Size and CPU

There are certain things you need to do if you are planning a replication scheme.

- The transaction log setting for `LogBufMB` 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 transaction log records from the file system. If the value of `LOG_FS_READS` is increasing, then increase the transaction 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 transaction log independently and needs adequate CPU resources to make progress.
- If the sending side and receiving side of the replication scheme are mismatched in CPU power, place the replication receiver on the faster system.

Performance Considerations When Altering Tables That Are Replicated

Altering a table to add or remove columns may lead to performance degradation or poor space utilization.

- When you alter a table to add one or more columns, the table is allocated a new partition for the additional columns. The additional partition causes extra processing when retrieving the data, resulting in reduced performance. See ALTER TABLE in the *Oracle TimesTen In-Memory Database SQL Reference*.
- When you alter a table to drop a column, the space is not always freed resulting in poor space utilization.

Any replication scheme defined with the `TABLE DEFINITION CHECKING EXACT` attribute requires that the physical structure of the table be identical on both master databases in order to be able to replicate operations between them. When using the `EXACT` table definition checking attribute, the only method to free the extraneous space resulted from dropped columns or eliminate extra partitions resulting from added columns is to drop and re-create the table, and then reload the data into the table.

However, if you create the tables with the `TABLE DEFINITION CHECKING RELAXED` attribute, then (while they must have the same key definition, number of columns, and column data types) the physical structure does not need to be identical on both master databases. The `TABLE DEFINITION CHECKING RELAXED` attribute can result in slightly slower performance, but only if the tables on both masters are not identical. The change in performance depends on the workload and the number of partitions and columns in the tables.

To improve performance for databases set with `RELAXED`, you can use `ttMigrate -r -relaxedUpgrade` to coalesce tables eliminating extraneous space from dropped columns or multiple partitions that were created when adding columns. This can be performed on one database, while the other database is still up and accepting requests on behalf of the application. You do not have to take both databases involved in replication down at the same time, but can perform `ttMigrate -r -relaxedUpgrade` on each one individually one after the other. This is optimal for databases where the tables are altered often and where the database can only perform online upgrades.

You can only coalesce partitions and eliminate extraneous space with `ttMigrate -r -relaxedUpgrade` on replicated tables when the table definition checking to `RELAXED`. However, if your tables have been using the `EXACT` attribute, then you can temporarily set table definition checking to `RELAXED`, consolidate the partitions and space for your tables, and then reset it to `EXACT`.

For more information on the `TABLE DEFINITION CHECKING RELAXED` attribute, see Column Definition Options for Replicated Tables.

Note

You can check if the table has multiple partitions. For details, see the instructions in both the Understanding Partitions When Using ALTER TABLE section detailed within ALTER TABLE in the *Oracle TimesTen In-Memory Database SQL Reference* and Check Partition Counts for Tables in the *Oracle TimesTen In-Memory Database Monitoring and Troubleshooting Guide*.

Increase Replication Throughput for Active Standby Pairs

You can increase replication throughput for active standby pairs by 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. Parallel replication instantiates multiple threads to transmit updates from the source database to the target database and to apply these updates on the target database. See Configuring Parallel Replication.

Limit Replication Transmitters, Receivers, and XLA Readers

Replication and XLA operations have significant overhead with transaction logging. Replication scales best when there are a limited number of transmitters or receivers.

Check your replication topology and see if you can simplify it. Generally, XLA scales best when there are a limited number of readers. If your application has numerous readers, see if you can reduce the number.

Monitor XLA and replication to ensure they are reading from the transaction log buffer rather than from the file system. With a lot of concurrent updates, replication may not keep up. Updates are single-threaded at the subscriber. You can achieve better XLA throughput if the frequency of acknowledgements is reduced.

Estimate the number of readers and transmitters required by checking the values in the `LOG_FS_READS` and `LOG_BUFFER_WAITS` columns in the `SYS.MONITOR` table. The system updates this information each time a connection is made or released and each time a transaction is committed or rolled back.

Setting `LogFlushMethod=`**2** can improve performance of `RETURN TWOSAFE` replication operations and `RETURN RECEIPT` with `DURABLE TRANSMIT` operations.

14 Managing Database Failover and Recovery

There are methods for you to manage database failover and recovery when using replication schemes.

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 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 a few things.

- 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

The following sections describe different procedures for managing failover:

- Subscriber Failures
- Master Failures
- Automatic Catch-Up of a Failed Master Database
- Failures in Bidirectional Distributed Workload Schemes
- Network Failures
- Failures Involving Sequences

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 Disabling Return Service Blocking Manually.

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 Transaction Log Failure Threshold 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. 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 an integer value of '1' if the database is failed, or '0' if not failed.

> **Note**
>
> 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.

However, if you are using bidirectional replication scheme where each database serves as both master and subscriber and one of the subscribers fail, then an error condition may occur. For example, assuming that the masters and subscribers for the bidirectional replication scheme are defined as follows:

```
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";
```

- If the `eastds` subscriber fails, the `westds` master stops accumulating updates for this subscriber since it received a failure.
- When the `eastds` subscriber fails, the replication agent shuts down on `eastds`. However, the `eastds` master continues accumulating updates to propagate to its subscriber on `westds` unaware that the replication agent has shut down. These updates continue to accumulate past the defined `FAILTHRESHOLD` since the replication agent (who propagates the records to the subscriber as well as monitors the `FAILTHRESHOLD`) is down.

If `TT_REPLICATION_INVALID` is set to 1 on a subscriber or standby database, the replication agent shuts down due to the fact that the subscriber or standby is no longer receiving updates. If your database fails when involved in a bidirectional configuration for your replication scheme, then the replication agent is not running and the `FAILTHRESHOLD` is not honored. To resolve this situation, destroy the subscriber or standby database and recreate it:

1. Destroy the failed database (in this example, the `eastds` database).
2. Re-create the failed database by performing a `ttRepAdmin -duplicate` operation from the other master in the bidirectional replication scheme (in this example, the master on `westds`).

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 in a Classic Replication Scheme 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, you must recover the master database as described in Recovering a Failed Database.

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.

See Recovering a Failed Database.

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. 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 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.

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.

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 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. 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.

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. Automatic catch-up also applies to recovery of master databases in active standby pairs.

See Automatic Catch-Up of a Failed Master Database.

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 and Recovering a Failed Database from a C Program.

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 Establishing Return Service Failure and Recovery Policies. 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 reestablish `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.

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 are 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.

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
```

You are 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.

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.

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 runs the procedure shown by the 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
                 -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.

A

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 A-1 Required attributes
- Table A-2 Conditionally required attributes
- Table A-3 Optional attributes

Table A-1 Required Attributes

Name	Description	Default
MasterHosts	Lists host names that may contain master databases in an active standby pair scheme.	None

Table A-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
AppType	The database to which the application should link.	None
CacheConnect	Specifies whether the active standby pair replicates cache groups.	N
MasterVIP	A list of two virtual IP addresses that can be associated with the master databases.	None

Table A-2 (Cont.) Conditionally Required Attributes

Name	Description	Default
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 is to be used for virtual IP addresses on each host.	None
VIPNetMask	The netmask of the virtual IP addresses.	None

Table A-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
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. If the application fails within this threshold, the failure is considered a failure to start properly, and the application is restarted on another host. If the application is active past this threshold, it is considered stable. If it fails after being stable, the application is restarted on the same host.	600

Table A-3 (Cont.) Optional Attributes

Name	Description	Default
AutoRecover	Specifies whether an active database should be automatically recovered from a backup if both master databases fail.	No
`ClusterType`	Defaults to `Active`. This is the only valid value for TimesTen.	Active
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
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 A-1.

MasterHosts

This attribute lists the host names that can contain master databases in the active standby pair. The first host listed has the active master database and the second host has the standby master database when the cluster is initially started.

The following are exceptions to this designated order:

- If there are already active and standby master databases on specific nodes when the cluster is stopped, then the active and standby master 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 is to be configured with the active master database. The first host listed for `MasterHosts` is to be the standby master database.

You can specify more than two hosts as master hosts for advanced availablity. See Configuring Advanced Availability.

If you have only two master hosts configured, you should not define any virtual IP addresses. If more than two master hosts are listed, you should configure virtual IP addresses for internal TimesTen processes to manage replication in the situation of an unrecoverable failure of one of the hosts on which an active or standby database resides. See Configuring Advanced Availability.

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 master in the active standby pair. The second host listed becomes the initial standby master in the active standby pair. The hosts defined must be different than the hosts defined for the SubscriberHosts attribute.

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 A-2.

AppCheckCmd

This attribute specifies the full command line for running 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. See AppName.

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 running an application that checks the status of the application specified by AppName.

Examples

```
AppCheckCmd=/mycluster/reader/app_check.sh 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 the Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation.

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. See AppName.

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

```
AppCheckCmd=/mycluster/reader/app_start.sh 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. See AppName.

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

```
AppCheckCmd=/mycluster/reader/app_stop.sh 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 the Oracle Clusterware Clusterware Administration and Deployment Guide in the Oracle Database documentation.

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 database password.

Setting

Set `CacheConnect` as follows:

How the Attribute Is Represented	Setting
`CacheConnect`	A value of `Y` (yes) or `N` (no). Default is `N`.

MasterVIP

This attribute defines two virtual IP (VIP) addresses associated with the two master hosts that are used by the active standby pair.

If you configure more than two master hosts for advanced availability, you must also define two VIP addresses. However, these VIP addresses are used solely by internal TimesTen processes for managing the active standby pair in the case of an unrecoverable error on one of the master hosts. Use of these VIP addresses by any user or application can result in an error. See Configuring Advanced Availability.

Setting

Set `MasterVIP` as follows:

How the Attribute Is Represented	Setting
`MasterVIP`	A comma-separated list of two virtual IP addresses to the master hosts.

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.

The directory must be shared by all hosts in the cluster. The shared storage must be NFS or OCFS (Oracle Cluster File System).

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 Configuring for Recovery When Both Master Nodes Permanently Fail and When Failures Occur on Both Master Nodes 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 the active standby pair is configured with subscribers, this attribute is required. It has no default value. You should have at least as many host names listed as there are subscribers defined.

If you are using advanced availability for your subscribers, define additional hosts that can be instantiated to take the place of any subscriber host that may encounter an unrecoverable error. Also, when using advanced availability, configure VIP addresses for every current subscriber in use. For example, if you have three subscribers and two additional hosts that can be used for advanced availability, you should have three VIP addresses defined. See SubscriberVIP.

Setting

Set `SubscriberHosts` as follows:

How the Attribute Is Represented	Setting
`SubscriberHosts`	A comma-separated list of host names. The hosts defined must be different than the hosts defined for the MasterHosts attribute. If not using advanced availability, the order of the hosts defined determines 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. When using advanced availability, the first hosts listed are used for the subscriber hosts. The latter hosts defined are used for advanced availability.

SubscriberVIP

This attribute configures a list of the virtual IP (VIP) addresses associated with subscriber hosts that are used for advanced availability.

When you configure more hosts than there are subscribers (for advanced availability), you must also define VIP addresses (one for each subscriber). However, these VIP addresses are used solely by internal TimesTen processes for managing replication in the case of an unrecoverable error on one of the subscriber hosts. Use of these VIP addresses by any user or application can result in an error. See Configuring Advanced Availability.

Setting

Set `SubscriberVIP` as follows:

How the Attribute Is Represented	Setting
`SubscriberVIP`	One or more virtual IP addresses. You should define the same number of VIP addresses as the number of subscribers. 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.

The optional attributes are listed in Table A-3.

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.

See AppScriptTimeout.

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.

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

The `FailureThreshold` 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

The `MasterStoreAttribute` attribute indicates the desired replication scheme `STORE` attributes for the master databases.

The `STORE` attributes apply to both the active and standby databases. For more information on `STORE` attributes, see Setting STORE Attributes.

This attribute is not required when RepDDL is configured.

If this attribute is not set, the `STORE` attributes take their default values.

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 Configuring for Recovery When Both Master Nodes Permanently Fail and When Failures Occur on Both Master Nodes.

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 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 Configuring for Recovery When Both Master Nodes Permanently Fail and When Failures Occur on Both Master Nodes 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.

If no value is specified for this attribute, the active standby pair is configured with no return service.

See Using a 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.

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.

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).