Oracle TimesTen In-Memory Database Java Developer's and Reference Guide

Release 6.0

B25266-03

For last-minute updates, see the TimesTen release notes.

March 2006

Printed in the United States of America

Contents

3 Writing an XLA Event Handler

4 Distributed Transaction Processing: JTA

5 Application Tuning

8 JMS/XLA Reference

Glossary

Index

About this Guide

Oracle® TimesTen is a high-performance, in-memory data manager that supports the ODBC and JDBC interfaces. The examples and procedures in this guide use the JDBC interface.

This guide is for application developers who use and administer TimesTen JDBC and for system administrators who configure and manage the TimesTen Daemon. It provides:

- Background information to help you understand how TimesTen works.
- Step-by-step instruction and examples that show how to perform the most commonly needed tasks.

To work with this guide, you should understand how database systems work. You should also have knowledge of SQL (Structured Query Language) and JDBC (Java DataBase Connectivity). See “Background reading” on page 3. if you are not familiar with these interfaces.

TimesTen documentation

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

- The *Oracle TimesTen In-Memory Database Installation Guide* provides information needed to install and configure TimesTen on all supported platforms.
- The *Oracle TimesTen In-Memory Database Architectural Overview* provides a description of all the available features in TimesTen.
- The *Oracle TimesTen In-Memory Database Operations Guide* provides information on configuring TimesTen and using the ttIsql utility to manage a data store. This guide also provides a basic tutorial for TimesTen.
- The *Oracle TimesTen In-Memory Database C Developer's and Reference Guide* and the *Oracle TimesTen In-Memory Database Java Developer's and Reference Guide* provide information on how to use the full set of available features in TimesTen to develop and implement applications that use TimesTen.
- The *Oracle TimesTen In-Memory Database API and SQL Reference Guide* contains a complete reference to all TimesTen utilities, procedures, APIs and other features of TimesTen.
- The *TimesTen to TimesTen Replication Guide*. This guide is for application developers who use and administer TimesTen and for system administrators who configure and manage TimesTen Replication. It provides: Background information to help you understand how TimesTen Replication works. Step-by-step instruction and examples that show how to perform the most commonly needed tasks.
- The *TimesTen Cache Connect to Oracle Guide* describes how to use Oracle Connect to cache Oracle data in TimesTen. This guide is for developers who use and administer TimesTen for caching Oracle data. It provides information on caching Oracle data in TimesTen data stores. It also describes how to use the Oracle Connect Administrator, a web-based interface for creating cache groups.

TimesTen documentation is available on the product CD-ROM and on the TimesTen web site: `http://www.timesten.com`.

Background reading

For a conceptual overview and JDBC development information, see:

- Hamilton, Cattell, Fisher. *JDBC Database Access with Java*. Reading, MA: Addison Wesley. 1998.

For a Java reference, see:

- Horstmann, Cornell. *Core Java*. Palo Alto, CA: Sun Microsystems Press. 1999.
- For the JDBC API specification, refer to java.sql package in the appropriate Java Platform API Specification.
  - If you are working with JDK 1.4, refer to the Java 2 Platform API specification at: `http://java.sun.com/j2se/1.4.2/docs/api/index.html`
  - If you are working with JDK 5.0, refer to the Java 2 Platform API specification at: `http://java.sun.com/j2se/1.5.0/docs/api/index.html`
- Siple, Matthew. *The Complete Guide to Java Database Programming: JDBC, ODBC and SQL*. McGraw-Hill. 1997.

For a review of SQL, see:

- Melton, Jim and Simon, Alan R. *Understanding the New SQL: A Complete Guide*. San Francisco, CA: Morgan Kaufmann Publishers. 1993.
- Groff, James R. / Weinberg, Paul N. *SQL: The Complete Reference.* McGraw-Hill. /1999

For information on Unicode, see:

- The Unicode Consortium, *The Unicode Standard, Version 3.0*, Addison-Wesley, 2000.
- The Unicode Consortium Home Page at `http://www.unicode.org`

Installing TimesTen

TimesTen Release 6.0 includes the TimesTen Data Manager for 32-bit and 64-bit platforms. See the *Oracle TimesTen In-Memory Database Installation Guide* for a description of supported platforms.

In addition to the Data Manager, TimesTen Release 6.0 also includes TimesTen Client and Server components. You can install the TimesTen Data Manager stand-alone or in a client/server environment.

For a list of the The TimesTen default installation directories, see the *Oracle TimesTen In-Memory Database Installation Guide*.

Conventions used in this guide

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

TimesTen documentation uses these typographical conventions:

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

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

If you see...	It means...
`fixed width italics`	Variable; must be replaced
`[ ]`	Square brackets indicate that an item in a command line is optional.
`{ }`	Curly braces indicated that you must choose one of the items separated by a vertical bar (|) in a command line.
`|`	A vertical bar (or pipe) separates arguments that you may use more than one argument on a single command line.
`...`	An ellipsis (. . .) after an argument indicates that you may use more than one argument on a single command line.
`%`	The percent sign indicates the UNIX shell prompt.
`#`	The number (or pound) sign indicates the UNIX root prompt.

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

If you see...	It means...
install_dir	The path that represents the directory where the current release of TimesTen is installed.
TTinstance	The instance name for your specific installation of TimesTen. Each installation of TimesTen must be identified at install time with a unique alphanumeric instance name. This name appears in the install path. The instance name "giraffe" is used in examples in this guide.
bits or *bb*	Two digits, either 32 or 64, that represent either the 32-bit or 64-bit operating system.
release or *rr*	Two digits that represent the first two digits of the current TimesTen release number, with or without a dot. For example, 51 or 5.0 represents TimesTen Release 5.0.
jdk_version	Two digits that represent the version number of the major JDK release. For example 14 for versions of jdk1.4.
timesten	A sample name for the TimesTen instance administrator. You can use any legal user name as the TimesTen administrator. On Windows, the TimesTen instance administrator must be a member of the Administrators group. Each TimesTen instance can have a unique instance administrator name.
DSN	The data source name.

Finding the information you need

The table below provides a brief overview of the TimesTen development process. It will help you get started with your application and find relevant information as you progress.

To learn how to	See
Install TimesTen.	*Oracle TimesTen In-Memory Database Installation Guide*
Create a data store and establish and close a connection.	Chapter 2, "Creating TimesTen Data Stores" and Chapter 3, "Working with the TimesTen Client and Server." in the *Oracle TimesTen In-Memory Database Operations Guide*.
Connect to a data store using JDBC.	"Connecting to a TimesTen Data Store" on page 28.
Resize a data store.	"Specifying the size of a data store" in the *Oracle TimesTen In-Memory Database Operations Guide*.
Compile and run a TimesTen application.	"Compiling and Executing Java Applications" on page 16"
Set up the TimesTen Client and Server.	Chapter 3, "Working with the TimesTen Client and Server" in the *Oracle TimesTen In-Memory Database Operations Guide*.
Work with TimesTen tables, indexes and rows.	Chapter 6, "Working with Data in a TimesTen Data Store" in the *Oracle TimesTen In-Memory Database Operations Guide*.
Understand options regarding transaction semantics.	Chapter 7, "Transaction Management and Recovery" in the *Oracle TimesTen In-Memory Database Operations Guide*.
View and change the query optimizer plan.	Chapter 9, "The TimesTen Query Optimizer" in the *Oracle TimesTen In-Memory Database Operations Guide*.
Use XLA to monitor changes to a data store	Chapter 3, "Writing an XLA Event Handler"
Replicate your system.	The *TimesTen to TimesTen Replication Guide*.

To learn how to	See
Achieve optimum performance.	Chapter 5, "Application Tuning" and Chapter 8, "Data Store Performance Tuning" in the *Oracle TimesTen In-Memory Database Operations Guide*.
Find troubleshooting information.	The *TimesTen Troubleshooting Procedures Guide*: http://www.timesten.com/support/docs/
Use TimesTen procedures.	The *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.
Use TimesTen utilities.	The *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

Technical Support

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

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

Email: timesten-support_us@oracle.com

Part I: TimesTen Java Developers' Guide

1

Configuring the Java Development Environment

This chapter describes how to install, configure, and test your TimesTen application development environment.

Topics in this chapter are:

- Installing TimesTen and the JDK
- Setting the Java Environment Variables
- Compiling and Executing Java Applications
- About the TimesTen Java Demos

Installing TimesTen and the JDK

Install and configure TimesTen for your environment, as described in the *Oracle TimesTen In-Memory Database Installation Guide* and the Java JDK, as described in your Java installation guide. The topics of particular interest in the *Oracle TimesTen In-Memory Database Installation Guide* when setting up a Java development environment are:

- "Access Control and non-root installations"
- "JDK support"
- "Client/Server configurations"
- "Environment modifications"

Once you have installed and configured TimesTen, create a data store DSN, as described in the *Oracle TimesTen In-Memory Database Operations Guide*. The topics of particular interest are:

- "TimesTen JDBC driver"
- "User and system DSNs"
- "Data Manager and Client DSNs"
- "Thread programming with TimesTen"
- "Creating a DSN on UNIX" or "Creating a DSN on Windows"

Setting the Java Environment Variables

The environment variable settings for TimesTen are explained in "Environment modifications" in the *Oracle TimesTen In-Memory Database Installation Guide*. This section provides more detail on those that impact the environment for TimesTen Java applications.

On UNIX platforms, you can set all of the environment variables described in this section by sourcing one of the following scripts:

install_dir/bin/ttSetEnv.sh

install_dir/bin/ttSetEnv.csh

On Windows, you can either set the environment variables during installation or run:

install_dir\bin\ttenv.bat

The rest of this section describes values the environment variables are set to, as well as how to set them manually, if necessary.

Set the CLASSPATH

Java classes and class libraries are found on CLASSPATH. Before executing a Java program that loads any of the TimesTen JDBC drivers, the `CLASSPATH` environment variable must contain the class library file:

install_dir/lib/classes*jdk_version*.jar

where *jdk_version* is a two-digit value indicating the first two digits of the version of the JDK that you are using (for example, for JDK 1.4, *jdk_version* would be 14).

Note: If more than one jar file is listed in the CLASSPATH, make sure the TimesTen jar file is listed first.

On UNIX, CLASSPATH elements are separated by colon. For example:

```
set CLASSPATH .:/opt/TimesTen/tt60/lib/classes14.jar
```

or

```
setenv CLASSPATH .:/opt/TimesTen/tt60/lib/classes14.jar
```

On Windows, CLASSPATH elements are separated by semicolons. Also, on Windows, do not use quotes when setting the CLASSPATH environment variable even if a directory pathname contains spaces.

For example, this is correct:

```
set CLASSPATH=.;C:/TimesTen/tt60/lib/classes14.jar
```

This is incorrect:

```
set CLASSPATH=.;"C:/TimesTen/tt60/lib/classes14.jar"
```

If in doubt about the JDK version you have installed on your system, use:

```
> java -version
```

If you are going to make use of the JMS/XLA interface described in Chapter 3, "Writing an XLA Event Handler, then you also need to add the following to your CLASSPATH:

```
install_dir/lib/timestenjmsxla.jar
install_dir/3rdparty/jms1.1/lib/jms.jar
```

For example, your CLASSPATH would look like:

```
.:C:/TimesTen/tt60/lib/classes14.jar:C:/TimesTen/tt60/lib/
timestenjmsxla.jar:C:/TimesTen/tt60/3rdparty/jms1.1/lib/jms.jar
```

Set the shared library path variable

Before running a java program that loads the TimesTen JDBC driver, the `shared library path for your system` environment variable must be set to include the TimesTen *`install_dir`*`/lib directory`. The name of the variable used for the shared library path depends on the system used:

System	Name of Variable
Linux	LD_LIBRARY_PATH
Solaris	LD_LIBRARY_PATH
HPUX	SHLIB_PATH or LD_LIBRARY_PATH
AIX	LIBPATH
Windows	PATH

See "Shared library path environment variable" in the *Oracle TimesTen In-Memory Database Installation Guide* for details on setting the shared library path.

Set the THREADS_FLAG variable (UNIX only)

The TimesTen JDBC driver uses native threads; green threads are not supported.

On some UNIX platforms, in order to use the native threads package, you must set the environment variable THREADS_FLAG to native. How you set the flag depends on your shell.

In csh, the syntax is:

```
setenv THREADS_FLAG native
```

In sh, the syntax is:

```
THREADS_FLAG=native
export THREADS_FLAG
```

Set the PATH variable

Make sure the executables **javac** and **java** are both on your executable search path, or will need to invoke them using absolute paths.

Compiling and Executing Java Applications

To compile a Java program, at your shell or command prompt use the command:

`javac` *`SourceFile`*`.java`

The command generates the bytecode file *`SourceFile`*`.class` if the `.java` file contains a public class. A `.class` file is generated for all classes defined in *`SourceFile`*`.java`. By default the `.class` files reside in the same directory as the `.java` source files. To specify a different target directory for the `.class` files, use the command:

`javac -d` *`Directory SourceFile`*`.java`

The class name is the same as the filename prefix of its corresponding `.class` file. To execute a Java program, at your shell or command prompt use the command:

`java` *`ClassName`*

`ClassName` is the name of a class that contains a `main` method. This command starts the Java Virtual Machine (JVM) that will interpret and execute the Java bytecode in the `.class` file and any other bytecode files that it is dependent upon.

Example 1.1 To compile the `level1.java` demo and execute it using the *demo* data store, enter:

```
> cd install_dir/demo/tutorial/java
> javac level1.java
> ttIsql -f ../datfiles/input0.dat demo
> java level1 demo
```

About the TimesTen Java Demos

Once you have configured your Java environment, you can confirm that everything is set up correctly by compiling and running the TimesTen Java demo applications in the *install_dir*/demo/tutorial/java directory.

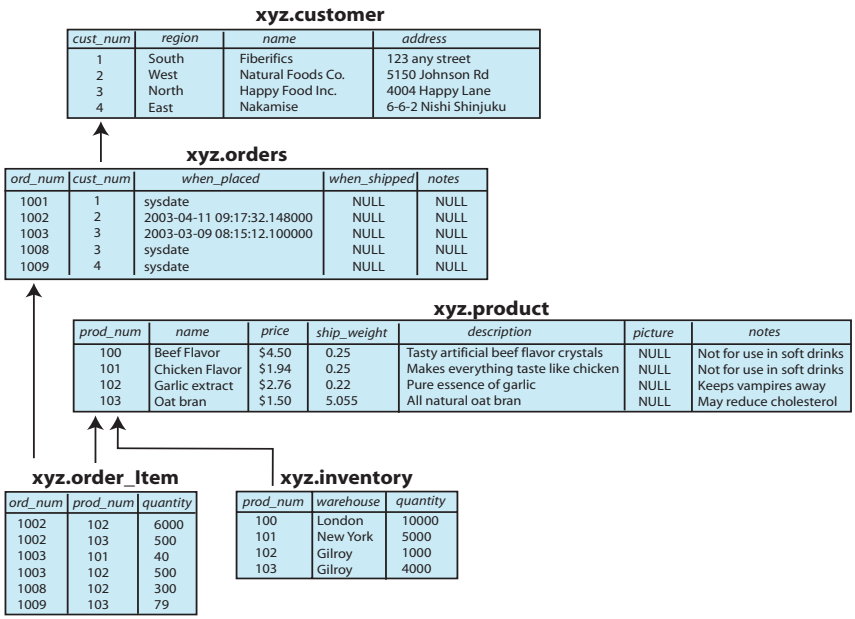
About the TimesTen Demo Schema

The TimesTen Java demos are designed to work with the TimesTen demo schema, which simulates a simple order-processing database. You can populate a data store with the TimesTen demo schema by running the *install_dir*/demo/tutorial/datfiles/input0.dat, as described in "Executing the TimesTen Java demos" on page 20. The input0.dat file creates the following tables:

- xyz.product
- xyz.inventory
- xyz.customer
- xyz.orders
- xyz.order_item

The tables in the demo schema are organized and populated with data, as shown in Figure 1.1

Figure 1.1 TimesTen Demo Schema



What the TimesTen demos do

The TimesTen Java demos are named `level1.java`, `level2.java`, `level3.java`, `level4.java`, `XlaLevel1.java`, `XlaLevel2.java`, and `XlaLevel3.java`. All of the `level` demos support both direct and client connections to the data store.

The `level1` demo uses the **DriverManager** interface to connect to a data store and forms a prepared INSERT and SELECT statement to insert new customer data into the `xyz.customer` table and then view the contents of the table. It executes the INSERT until all of the data in the `input1.dat` file is loaded into the table, executes the SELECT, fetches and prints the result set to `stdout` and disconnects from the data store.

The `level2` demo uses the **TimesTenDataSource** interface to connect to a data store and forms prepared INSERT, UPDATE, DELETE, and SELECT statements to insert, update, delete, and view product data in the xyz.product table. It executes the INSERT until all of the data in the `input2.dat` file is loaded into the table. It executes the DELETE to delete any duplicate product data and then the UPDATE to increase the price of the products in the table by 10%. It executes a **ttCkpt** procedure to checkpoint the data store to disk, executes the SELECT, fetches and prints the result set to `stdout` and then disconnects from the data store.

The `level3` demo uses the **TimesTenDataSource** interface to connect to a data store and forms prepared statements to perform order processing operations on the order data in the `input3.dat` file. For each order item in the data file, the demo performs the following transaction:

- INSERT the new order into the xyz.orders and `xyz.order_item` tables.
- SELECT from the `xyz.inventory` table to check the available quantity of the ordered item in the inventory.
- UPDATE the `xyz.inventory` table to debit the ordered item from the inventory.

If there are not enough items in the inventory, the demo rolls back the entire transaction and reports that there is insufficient inventory for the order. Finally, the demo checkpoints the data store to disk and disconnects.

The `level4` demo processes the same orders as `level3`, only it uses multiple threads and multiple connections to increase throughput.

Both the `level1` and `level2` demos call the TimesTen **ttOptUpdateStats** built-in procedure to update the statistics for the `customer` and `product` tables. The **ttOptUpdateStats** procedure stores the statistics in the SYS.COL_STATS and SYS.TBL_STATS tables for use by the TimesTen query optimizer to enable more efficient query execution.

The `XlaLevel1.java`, `XlaLevel2.java`, and `XlaLevel3.java` demos use the JMS/XLA API described in Chapter 3, "Writing an XLA Event Handler to monitor and report on specific updates to the data store. The `XlaLevel1.java` and `XlaLevel2.java` demos monitor updates to the `xyz.customer` table. The `XlaLevel3.java` demo monitors updates to a user-specified table.

Compiling the TimesTen Java demos

To compile the Java demos, go to the Java demo directory and run ANT on the `build.xml` file. If you don't want to use ANT, use the `javac` command to compile each demo. For example:

```
> cd /TimesTen/tt60/demo/tutorial/java
> javac *.java
```

Executing the TimesTen Java demos

Prior to executing any of the Java demos, you must execute the SQL statements in the `input0.dat` file, as shown below, to build or rebuild the demo schema. Each demo requires that you specify a DSN name. The DSN can be for either a direct connection or client connection to the data store.

Executing the level demos

All of the level demos have the following command syntax:

```
demoname [-t] [-d | -c] {DSN}
demoname -h | -help

  -h | -help    print usage and exit
  -d            connect using direct driver (default)
  -c            connect using client driver
  -t            enable JDBC tracing
  DSN           name of the data store
```

Example 1.2 The various ways an application can connect to a data store are described in Chapter 4, "How Applications Connect to Data Stores" in the *Oracle TimesTen In-Memory Database Architectural Overview*. In this example, we execute the `level1` and `level2` demos using a *direct driver* connection to the *DMdemo* data store. We enable JDBC tracing when executing the `level2` demo. Last, we execute the `level3` demo using a client connection to the *CSdemo* data store.

Note: Before executing each demo, you must execute the `input0.dat` file to rebuild the demo schema on the data store.

```
> ttIsql -f ../datfiles/input0.dat DMdemo
...... output
> java level1 DMdemo
...... output
> ttIsql -f ../datfiles/input0.dat DMdemo
...... output
> java level2 -t DMdemo
...... output
> ttIsqlCS -f ../datfiles/input0.dat CSdemo
...... output
> java level3 -c CSdemo
...... output
```

If you cannot connect to the data store, you may not have configured the DSN name that you are specifying. See the "Data source names" section in the *Oracle TimesTen In-Memory Database Operations Guide*.

Executing the XlaLevel demos

Note: You only need to look at the XlaLevel demos if you are going to be using the JMS/XLA API described in Chapter 3, "Writing an XLA Event Handler."

In JMS/XLA, the name of the data store and other parameters used by XLA applications are specified in the form of *XLA topics*, as described in "XLA topics" on page 55. The XlaLevel demos obtain their XLA topics from the `jmsxla.xml` file located in the *`install_dir`*`/demo/tutorial/java` directory. The topics in the `jmsxla.xml` file are configured so that the XlaLevel demos use the predefined **RunData_*TTinstance*** data store and other default parameters. You can edit the `jmsxla.xml` file to specify other topics or change the settings in the exiting topics.

Note: The demos in this document use the predefined data store **RunData_tt60**.

All of the XlaLevel demos accept an optional topic name.

The syntax for executing `XlaLevel1` is:

XlaLevel1 [*topic*]

where *topic* defaults to the *Level1Demo* topic that is prespecified in the `jmsxla.xml` file.

The syntax for executing `XlaLevel2` is:

XlaLevel2 [*topic* [*bookmark*]]

where *topic* defaults to *Level2Demo* and *bookmark* defaults to *bookmark*.

The syntax for executing `XlaLevel3` is:

XlaLevel3 [*topic* [*bookmark* [*table*]]]

where *topic* defaults to *Level3Demo*, *bookmark* defaults to *bookmark*, and *table* defaults to *tbl*.

Prior to executing any of the `XlaLevel1` and `XlaLevel2` demos, you must execute the SQL statements in the `input0.dat` file to build or rebuild the demo schema. Both the `XlaLevel1` and `XlaLevel2` demos monitor changes to the `xyz.customer` table.

When running the XlaLevel1 demo, you can make updates to the xyz.customer table by running the level1.java demo in a separate shell.

Example 1.3 To run the XlaLevel1.java demo with its default topic, in one shell enter:

```
> ttIsql -f ../datfiles/input0.dat RunData_tt60
...... output
> java XlaLevel1
...... detected changes to the xyz.customer table
```

In another shell, enter:

```
> java level1 RunData_tt60
...... output
```

The output from the XlaLevel1 demo shows the detected changes to the xyz.customer table.

If you create a new topic in the jmsxla.xml file, you can specify that when you enter run the XlaLevel1.java demo. For example, if you created a new topic, named *MyTopic*, you would start the XlaLevel1.java demo with:

```
> java XlaLevel1 MyTopic
```

The XlaLevel2 demo prompts you to enter changes to the xyz.customer table in the form of SQL from the command line. It then displays the detected changes to the table after you commit the transaction.

Example 1.4 To run the XlaLevel2.java demo with its default topic and bookmark, enter:

```
> ttIsql -f ../datfiles/input0.dat RunData_tt60
...... output
> java XlaLevel2
+++ Using default topic Level2Demo and default bookmark bookmark
+++ create session
+++ create topic
+++ createDurableSubscriber
+++ using connection string
'DSN=RunData_tt60;ExclAccess=0;Logging=1;LogPurge=0;Overwrite=0'
+++ connecting to
jdbc:timesten:direct:DSN=RunData_tt60;ExclAccess=0;Logging=1;LogPurge=0;Overwrit
e=0
+++ turning off autocommit
You can now enter SQL commands.  You should enter
either DML (such as inserts, updates, or deletes),
or DDL (such as CREATE SEQUENCE).
```

```
For instance, try:
  create sequence s minvalue 1000
  insert into xyz.customer values(s.nextVal,'us','Bob','nowhere')
  commit
After each SQL command you enter, the demo tries
to get and display any JMS/XLA updates.
Type "quit" to exit the demo, or "help"
to see this message again.
NOTE: autocommit is turned off, so you will
have to enter "commit" to see your updates.
Enter SQL: create sequence s minvalue 1000
+++ create sequence s minvalue 1000
Enter SQL: insert into xyz.customer values(s.nextVal,'us','Bob','nowhere')
+++ insert into xyz.customer values(s.nextVal,'us','Bob','nowhere')
Enter SQL: insert into xyz.customer values(s.nextVal,'us','Bob','nowhere')
+++ insert into xyz.customer values(s.nextVal,'us','Bob','nowhere')
Enter SQL: commit
+++ commit
>>> got a CREATE SEQUENCE message
  CYCLE=true  INCREMENT=1  MAX_VALUE=9223372036854775807  MIN_VALUE=1000
  NAME=S  OWNER=ASPIN  __COMMIT=false  __CONTEXT=(null)  __FIRST=true
  __REPL=false  __TYPE=16  __mtyp=null  __mver=1144080
>>> got a INSERT message
  ADDRESS=nowhere  CUST_NUM=1000  NAME=Bob  REGION=us     __NULLS=
  __TYPE=10  __mtyp=null  __mver=1146360
>>> got a INSERT message
  ADDRESS=nowhere  CUST_NUM=1001  NAME=Bob  REGION=us     __COMMIT=true
  __NULLS=  __TYPE=10  __mtyp=null  __mver=1147248
Enter SQL: quit
+++ cleaning up
+++ Subscriber close
+++ Producer.close
+++ done
+++ shutting down...
```

The `XlaLevel3` demo prompts you to specify the table you wish to make changes to and to monitor. If the table doesn't exist in the *RunData_tt60* data store, it is created.

Example 1.5 To run the `XlaLevel3.java` demo with its default topic, *Level3Demo*; a bookmark named *bkmk*, and to create a new table, named *tbl*, enter:

```
> java XlaLevel3 Level3Demo bkmk tbl
+++ topic=Level3Demo, bookmark=bkmk, table=tbl
May 11, 2005 3:32:26 PM
com.timesten.dataserver.jmsxla.SimpleInitialContextFactory getInitialContext
INFO: Using configuration file jmsxla.xml
+++ create session
+++ create topic
May 11, 2005 3:32:27 PM com.timesten.dataserver.jmsxla.DestinationImpl <init>
FINE: Properties for topic Level3Demo:{xlaPrefetch=100, name=Level3Demo,
connectionString=DSN=RunData_tt60}
+++ createDurableSubscriber
May 11, 2005 3:32:27 PM com.timesten.dataserver.jmsxla.XlaSubscriber <init>
FINE: Making XLA subscription, connstr=DSN=RunData_tt60, bookmark=bkmk,
prefetch=100, ackMode=1 May 11, 2005 3:32:27 PM
com.timesten.dataserver.jmsxla.MessageConsumerImpl createXlaSubscriber
FINE: Creating MessageConsumer with connection string=DSN=RunData_tt60,
bookmark=bkmk May 11, 2005 3:32:27 PM
com.timesten.dataserver.jmsxla.XlaSubscriber start
FINE: Starting XLA subscription
+++ using connection string 'DSN=RunData_tt60'
+++ connecting to jdbc:timesten:direct:DSN=RunData_tt60
+++ turning off autocommit
table tbl already exists
+++ {call ttXlaSubscribe('tbl', 'bkmk')}
You can now enter SQL commands.  You should enter either DML (such as inserts,
updates, or deletes), or DDL (such as CREATE SEQUENCE).
For instance, try:
  create sequence s minvalue 1000
  insert into tbl values(s.nextVal)
  call ttApplicationContext('inserted something')
  commit
After each SQL command you enter, the demo tries to get and display any JMS/XLA
updates.
Type "quit" to exit the demo, or "help"
to see this message again.
NOTE: autocommit is turned off, so you will have to enter "commit" to see your
updates.
Enter SQL: create sequence s minvalue 1000
+++ create sequence s minvalue 1000
Enter SQL: insert into tbl values(s.nextVal)
+++ insert into tbl values(s.nextVal)
```

```
Enter SQL: call ttApplicationContext('inserted something')
+++ call ttApplicationContext('inserted something')
Enter SQL: commit
+++ commit
>>> got a CREATE TABLE message
  NAME=TBL  OWNER=ASPIN  _A_INPRIMARYKEY=null  _A_NULLABLE=null
  _A_OUTOFLINE=null  _A_PRECISION=null  _A_SCALE=null  _A_SIZE=null
  __COMMIT=null  __FIRST=null  __TYPE=null  __mtyp=null  __mver=null
>>> got a CREATE SEQUENCE message
  CYCLE=true  INCREMENT=1  MAX_VALUE=9223372036854775807  MIN_VALUE=1000
  NAME=S  OWNER=ASPIN  __COMMIT=false  __CONTEXT=(null)  __FIRST=true
  __REPL=false  __TYPE=16  __mtyp=null  __mver=385368
>>> got a INSERT message
  A=1000  __NULLS=B  __TYPE=10  __mtyp=null  __mver=386576
>>> got a COMMIT ONLY message
  __COMMIT=true  __CONTEXT=inserted something  __FIRST=false
  __REPL=false  __TYPE=13  __mtyp=null  __mver=386880 Enter SQL: create index ix
on tbl(a)
+++ create index ix on tbl(a)
Enter SQL: commit
+++ commit
>>> got a CREATE INDEX message
  COLUMNS=A  HASH_PAGES=0  INDEX_METHOD=T  INDEX_TYPE=R  IXNAME=IX
  TBLNAME=TBL  TBLOWNER=ASPIN  UNIQUE=false  __COMMIT=true
  __CONTEXT=(null)  __FIRST=true  __REPL=false  __TYPE=3  __mtyp=null
  __mver=392904
Enter SQL: drop index ix
+++ drop index ix
Enter SQL: insert into tbl values(s.nextVal)
+++ insert into tbl values(s.nextVal)
Enter SQL: insert into tbl values(s.nextVal)
+++ insert into tbl values(s.nextVal)
Enter SQL: update tbl set a=a+10
+++ update tbl set a=a+10
Enter SQL: commit
+++ commit
>>> got a DROP INDEX message
  INDEX_NAME=IX  OWNER=ASPIN  TABLE_NAME=TBL  __COMMIT=false
  __CONTEXT=(null)  __FIRST=true  __REPL=false  __TYPE=4  __mtyp=null
  __mver=398776
>>> got a INSERT message
  A=1001  __NULLS=B  __TYPE=10  __mtyp=null  __mver=399472
>>> got a INSERT message
  A=1002  __NULLS=B  __TYPE=10  __mtyp=null  __mver=400112
>>> got a UPDATE message
  A=1010  _A=1000  __NULLS=_B;B  __TYPE=11  __UPDCOLS=A  __mtyp=null
  __mver=400712
```

```
>>> got a UPDATE message
  A=1011  _A=1001  __NULLS=_B;B  __TYPE=11  __UPDCOLS=A  __mtyp=null
  __mver=401256
>>> got a UPDATE message
  A=1012  _A=1002  __COMMIT=true  __NULLS=_B;B  __TYPE=11  __UPDCOLS=A
  __mtyp=null  __mver=401800
Enter SQL: quit
+++ cleaning up
+++ Subscriber close
May 11, 2005 3:35:04 PM com.timesten.dataserver.jmsxla.XlaSubscriber
tableUnsubscribe
FINE: Unsubscribing from table TBL
+++ Producer.close
+++ done
+++ shutting down...
```

Problems executing the TimesTen Java demo programs

If you receive an error message like:

```
java.lang.UnsatisfiedLinkError:no ttJdbcCS
```

or

```
java.lang.UnsatisfiedLinkError: no ttJdbc in java.library.path
```

then you don't have LD_LIBRARY_PATH set properly; find `libttJdbc.so` and put that directory on the LD_LIBRARY_PATH:

```
setenv LD_LIBRARY_PATH install_dir/lib
```

Problems compiling the TimesTen Java demo program

If you receive the error message:

```
java.lang.ClassNotFoundException:com.timesten.jdbc.TimesTenDriver
```

CLASSPATH is not set properly. Find the classes archive file and make sure that it is on the CLASSPATH, for example:

```
setenv CLASSPATH install_dir/lib/classes14.jar
```

If you get a ClassNotFoundException for a class defined in one of the demos (such as ‘level1’), make sure the current directory is included in your CLASSPATH. For example:

```
setenv CLASSPATH install_dir/lib/classes14.jar:.
```

2

Working with TimesTen Data Stores

This chapter describes the basic procedures for writing a Java application to access data in a TimesTen data store. Before attempting to write a TimesTen application, be sure you have completed the following prerequisite tasks:

Prerequisite Task	What you do
Create a TimesTen data store	Follow the procedures described in Chapter 2, "Creating TimesTen Data Stores" in the *Oracle TimesTen In-Memory Database Operations Guide*
Configure Java environment	Follow the procedures described in "Setting the Java Environment Variables" on page 14
Compile and execute the TimesTen Java demos	Follow the procedures described in "About the TimesTen Java Demos" on page 17

Once you have successfully executed the TimesTen Java demos, your development environment is set up correctly and ready for you to create applications that accesses a TimesTen data store.

Topics in this chapter are:

Java Classes

Most TimesTen applications can be written using the supported Java classes and interfaces listed in Chapter 7, "JDBC Reference."

You must import the standard JDBC packages in any java program that use JDBC:

```
import java.sql.*;
```

If you are going to make use of the **DataSource** interface, you must also import the optional JDBC packages:

```
import javax.sql.*;
```

Though you can accomplish most operations with the standard Java interfaces, TimesTen provides the following extensions to the Java standard.

To use the TimesTen implementation of the **javax.sql.DataSource** interface, import:

```
import com.timesten.jdbc.TimesTenDataSource;
```

To use the TimesTen implementation of the **javax.sql.XADataSource** interface, import:

```
import com.timesten.jdbc.xa.TimesTenXADataSource;
```

To use the TimesTen connection-based prefetch feature described in "Fetching multiple rows of data" on page 39, import:

```
import com.timesten.sql.TimesTenConnection;
```

See "TimesTen Extensions to JDBC" on page 98 for more information on these TimesTen extentions.

Connecting to a TimesTen Data Store

The different ways an application can connect to a data store are described in Chapter 4, "How Applications Connect to Data Stores" in the *Oracle TimesTen In-Memory Database Architectural Overview*. How to create a DSN to define a connection to a TimesTen data store is described in the *Oracle TimesTen In-Memory Database Operations Guide*.

The type of DSN you create depends on whether your application is to connect directly to the data store, or by means of client connection. If you intend to connect directly to the data store, create a DSN as described in "Creating a DSN on UNIX" or "Creating a DSN on Windows" in the *Oracle TimesTen In-Memory Database Operations Guide*. If you intend to create a client connection to the data store, create a DSN as described in "Creating and configuring Client DSNs on Windows" or "Creating and configuring Client DSNs on UNIX" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Once you have created a DSN, your application can connect to the data store. This section describes how to create a JDBC connection to a data store using either the JDBC direct driver or the JDBC client driver.

The operations described in this section are based on the `level1.java` demo.

The procedures for connecting to a TimesTen data store are:

- Load the TimesTen driver
- Create a connection URL for the data store
- Connect to the data store
- Disconnect from the data store
- Putting it all together -- preparing and executing SQL

Load the TimesTen driver

The TimesTen JDBC driver must be loaded before it is available for making connections with a TimesTen data store. The TimesTen JDBC driver is:

```
com.timesten.jdbc.TimesTenDriver
```

If you are using the **DriverManager** interface to connect to TimesTen, call the **Class.forName()** method to load the TimesTen JDBC driver. This method creates an instance of the TimesTen Driver and registers it with the driver manager.

If you are using the **TimesTenDataSource** interface, you do not need to call **Class.forName()**.

Example 2.1 To identify and load the TimesTen driver:

```
Class.forName("com.timesten.jdbc.TimesTenDriver");
```

Note: If the TimesTen JDBC driver is not loaded, an error is returned when your application attempts to connect to a TimesTen data store.

Create a connection URL for the data store

To create a JDBC connection, you need to specify a TimesTen connection URL for the data store. The format of a TimesTen connection URL is:

```
jdbc:timesten:{direct | client}:dsn=DSNname;[DSNattributes;]
```

For example, to create a direct connection to the *demo* data store, the URL would look like:

```
String URL = "jdbc:timesten:direct:dsn=demo";
```

Specifying data store attributes in the connection URL

You can programmatically set or override the connection attributes in the DSN description by specifying attributes in the connection URL.

For example, to set the **LockLevel** DSN attribute to '1', you could create a URL like:

```
String URL = "jdbc:timesten:direct:dsn=demo;LockLevel=1";
```

Note: When caching Oracle data in TimesTen, the OracleID attribute must be set in the TimesTen ODBC Setup dialog or the ODBC.INI file. To prevent confusion between connected clients, you should set the OracleID programmatically. For more information on caching Oracle data in TimesTen, see the *TimesTen Cache Connect to Oracle Guide*.

Connect to the data store

Once you have defined a URL, you can use either the **DriverManager.getConnection()** or **TimesTenDataSource**.**getConnection()** method to connect to the TimesTen data store.

If you use the **DriverManager.getConnection()** method, simply specify the driver URL to connect to the data store:

```
import java.sql.*;

Connection con = DriverManager.getConnection(URL);
```

To use the **TimesTenDataSource.getConnection()** method, first create a **TimesTenDataSource**, the use the **TimesTenDataSource.setUrl()** method to set the URL and **TimesTenDataSource.getConnection()** to connect:

```
import com.timesten.jdbc.TimesTenDataSource;

TimesTenDataSource ds = new TimesTenDataSource();
ds.setUrl(URL);
con = ds.getConnection();
```

Either method returns a **Connection** object (*con* in this example) that you can use as a handle to the data store. See the `level1` demo for an example on how to use **DriverManager.getConnection()** and the `level2` and `level3` demos for examples on how to use **TimesTenDataSource.getConnection()**.

Disconnect from the data store

When you are finished accessing the TimesTen data store, call the **Connection.close()** method to close the connection to the data store.

If an error has occurred, you may want to roll back the transaction before disconnecting from the data store. See "Handling non-fatal errors" on page 47 and "Rolling back failed transactions" on page 51 for more information.

Opening and closing a direct driver connection

Example 2.2 shows the general framework for an application that uses the **DriverManager** to create a direct driver connection to the *demo* data store; execute some SQL, and then close the connection. See the `level1.java` demo for a working example.

Example 2.2

```
String URL = "jdbc:timesten:dsn=demo";
Connection con = null;

try {
     Class.forName("com.timesten.jdbc.TimesTenDriver");
} catch (ClassNotFoundException ex) {
       // See "Handling Errors" on page 46
}

try {
     // Open a connection to TimesTen
     con = DriverManager.getConnection(URL);

     // Report any SQLWarnings on the connection
     // See "Reporting errors and warnings" on page 48

     // Do SQL operations
     // See "Managing TimesTen Data" on page 32

// Close the connection to TimesTen
con.close();

// Handle any errors
} catch (SQLException ex) {
     // See "Handling Errors" on page 46
}
```

Managing TimesTen Data

This section provides detailed information on working with data in a TimesTen data store.

Topics:

Calling SQL statements within Java applications

The main topics in this section are:

Setting auto commit

A TimesTen Connection has auto-commit enabled by default. You can use **Connection.setAutoCommit()** to enable or disable auto-commit. If auto-commit is disabled (set to false), you must use **Connection.commit()** to manually commit transactions.

For example, to set auto-commit to off:

```
con.setAutoCommit(false);
```

```
// Report any SQLWarnings on the connection
// See "Reporting errors and warnings" on page 48
```

Preparing SQL statements

SQL statements that are to be executed more than once should be prepared in advance by calling the **Connection.prepareStatement()** method.

For maximum performance, prepare parameterized statements. Example 2.3 shows how four separate INSERT statements can be substituted with a single parameterized statement.

Example 2.3 Rather than execute a similar INSERT statement with different values:

```
Statement.execute("insert into t1 values (1, 2)");
Statement.execute("insert into t1 values (3, 4)");
Statement.execute("insert into t1 values (5, 6)");
Statement.execute("insert into t1 values (7, 8)");
```

It is much more efficient to prepare a single parameterized INSERT statement and use **PreparedStatement.set...()** methods to set the row values before each execute:

```
PreparedStatement pIns =
       con.PreparedStatement("insert into t1 values (?,?)");
con.commit();

pIns.setInt(1, Integer.parseInt(1));
pIns.setInt(2, Integer.parseInt(2));
pIns.executeUpdate();

pIns.setInt(1, Integer.parseInt(3));
pIns.setInt(2, Integer.parseInt(4));
pIns.executeUpdate();

pIns.setInt(1, Integer.parseInt(5));
pIns.setInt(2, Integer.parseInt(6));
pIns.executeUpdate();

pIns.setInt(1, Integer.parseInt(7));
pIns.setInt(2, Integer.parseInt(8));
pIns.executeUpdate();

con.commit();
pIns.close();
```

Note: After preparing SQL statements, call **Connection.commit()** in order to release the locks held by the prepare and to allow the query plan to persist. When you have finished executing a prepared statement, call the **PreparedStatement.close()** method to release the resources associated with the statement.

TimesTen shares prepared statements automatically once they have been committed. For example, if two or more separate connections to the data store each prepare the same statement, then the 2nd, 3rd, and *n*th prepare returns very quickly because TimesTen remembers the first prepared statement.

Example 2.4 In this example, we prepare three identical parameterized INSERT statements for three separate connections. The first prepared INSERT for connection *con1* is shared with the *con2* and *con3* connections and speeds up the *pIns2* and *pIns3* prepare operations:

```
Connection con1;
Connection con2;
Connection con3;
.....
PreparedStatement pIns1 = con1.prepareStatement
                  ("insert into t1 values (?,?)");
con1.commit();

PreparedStatement pIns2 = con2.prepareStatement
                  ("insert into t1 values (?,?)");
con2.commit();

PreparedStatement pIns3 = con3.prepareStatement
                  ("insert into t1 values (?,?)");
con3.commit();
```

Note: All tuning options, such as join ordering, indexes and locks must match for the statement to be shared. Also, if the prepared statement references a temp table, it will only be shared within a single connection.

Note: TimesTen also supports prepared statement pooling for **PooledConnection**s, as specified in the JDBC 3.0 specification. You can configure the maximum size of the pool by setting **ObservableConnectionDS.setMaxStatements()**. Once set, this value should not be changed.

See "Prepare statements in advance" on page 81 for a general discussion of the performance benefits of preparing SQL statements in advance.

Executing SQL statements

Chapter 6, "Working with Data in a TimesTen Data Store" in the *Oracle TimesTen In-Memory Database Operations Guide* describes how to use SQL to manage data in a TimesTen data store. This section describes how to use the **Connection.createStatement()**, **Statement.executeUpdate()**, and **Statement.executeQuery()** methods to execute a SQL statement within a Java application.

Unless statements are prepared in advance, as described in "Preparing SQL statements", use the **Statement** execute methods, such as **Statement.execute(), Statement.executeUpdate()**, or **Statement.executeQuery()**, depending on the nature of your SQL statement and any returned result set.

For SQL statements that are a prepared in advance, use the **PreparedStatement** execute methods, such as **PreparedStatement.execute()**, **PreparedStatement.executeUpdate()**, or **PreparedStatement.executeQuery()**.

The **execute()** method returns True if there is a result set (for example, on. a SELECT) or False if there is no result set (for example, on an INSERT, UPDATE, or DELETE). The **executeUpdate()** method returns the number of rows affected. For example, when executing an INSERT statement, the **executeUpdate()** method returns the number of rows inserted. The **executeQuery()** method returns a result set, so it should only be called when a result set is expected (for example, when executing a SELECT).

Note: See "Working with result sets" on page 42 for details on what you need to know when working with result sets generated by TimesTen.

Example 2.5 to use the **Statement.executeUpdate()** method to execute an INSERT into the *xyz.customer* table, enter:

```
Connection con;
Statement stmt;
. . . . . .
try {
    stmt = con.createStatement();
    int numRows = stmt.executeUpdate("insert into xyz.customer
       values" + "(40, 'West', 'Big Dish', '123 Signal St.');");
}
catch (SQLException ex) {
    .....
}
```

Example 2.6 In this example, we use a **Statement.executeQuery()** method to execute a SELECT on the *xyz.customer* table and display the returned **ResultSet**:

```
Statement stmt;
. . . . . .
try {
  ResultSet rs = stmt.executeQuery("select cust_num, region, " +
                       "name, address from xyz.customer;");

  System.out.println("Fetching result set...");
  while (rs.next()) {
     System.out.println("\n  Customer number: " + rs.getInt(1));
     System.out.println("  Region: " + rs.getString(2));
     System.out.println("  Name: " + rs.getString(3));
     System.out.println("  Address: " + rs.getString(4));
     }
  }
catch (SQLException ex) {
    ex.printStackTrace();
}
```

Example 2.7 In this example, we use a **PreparedStatement.executeQuery()** method to execute a prepared SELECT statement and display the returned **ResultSet**:

```
PreparedStatement pSel = con.prepareStatement("select cust_num, " +
                         "region, name, address " +
                         "from xyz.customer;");

con.commit();


try {
  ResultSet rs = pSel.executeQuery();

  while (rs.next()) {
      System.out.println("\n  Customer number: " + rs.getInt(1));
      System.out.println("  Region: " + rs.getString(2));
      System.out.println("  Name: " + rs.getString(3));
      System.out.println("  Address: " + rs.getString(4));
    }
}
catch (SQLException ex) {
    ex.printStackTrace();
}
```

Setting a timeout value for executing SQL statements

In TimesTen you can set the DSN attribute **SqlQueryTimeout** to specify the query timeout period for all connections. If you set **SqlQueryTimeout** in the DSN specification, its value becomes the default value for all subsequent connections to the data store.

You can override the **SqlQueryTimeout** value for the current connection by calling the **Statement.setQueryTimeout()** method to set the time limit in seconds for which the data store should execute SQL queries. In TimesTen, once the timeout trigger fires it indicates to the executing query that it must timeout. Since there can be a lag in the time that it takes the timeout message to get to the query, the actual time it takes for the query to end is approximately the time it takes for the time out message to get to the query plus the timeout value specified.

The **Statement.setQueryTimeout()** method works only when the SQL statement is actively executing. A timeout does not occur during the commit or rollback phase of the operation. For those transactions that do a large number of UPDATEs DELETEs or INSERTs, the commit or rollback phases may take a long time to complete. During that time the timeout value is ignored.

Note: The **LockWait** and **SqlQueryTimeout** settings in TimesTen are separate features and can have separate values. TimesTen checks for both lock timeout and SQLQueryTimeout values. TimesTen examines all threads and wakes up any sleeping process that has either a lock timeout or a SQL query timeout, and indicates the appropriate **SqlQueryTimeout** value to any executing thread. If both a **LockWait** timeout value and a **SqlQueryTimeout** value are specified, the lesser of the two values causes a timeout first.

Putting it all together -- preparing and executing SQL

In this example, we prepare an INSERT and SELECT statement; execute the INSERT twice; execute the SELECT, and print the returned result set. For a working example, see the `level1.java` demo.

```
Connection con;
Statement stmt;

// Disable auto-commit
con.setAutoCommit(false);

     // Report any SQLWarnings on the connection
     // See "Reporting errors and warnings" on page 48

// Prepare a parameterized INSERT and a SELECT Statement
PreparedStatement pIns = con.prepareStatement("insert into
xyz.customer values (?,?,?,?)");

PreparedStatement pSel = con.prepareStatement
   ("select cust_num, region, name, " +
    "address from xyz.customer");

// Prepare is a transaction; must commit to release locks
con.commit();

// Data for first INSERT statement
pIns.setInt(1, Integer.parseInt(100));
pIns.setString(2, 'N');
pIns.setString(3, 'Fiberifics');
pIns.setString(4,'123 any street');

// Execute the INSERT statement
pIns.executeUpdate();

// Data for second INSERT statement
pIns.setInt(1, Integer.parseInt(101));
pIns.setString(2, 'N');
pIns.setString(3,'Natural Foods Co.');
pIns.setString(4,'5150 Johnson Rd');

// Execute the INSERT statement
pIns.executeUpdate();

// Commit the inserts
con.commit();

// Done with INSERTs, so close the prepared statement
pIns.close();
```

```
// Report any SQLWarnings on the connection
reportSQLWarnings(con.getWarnings());
CheckIfStopIsRequested();

// Execute the prepared SELECT statement
ResultSet rs = pSel.executeQuery();

System.out.println("Fetching result set...");
while (rs.next()) {
    System.out.println("\n Customer number: " + rs.getInt(1));
    System.out.println("  Region: " + rs.getString(2));
    System.out.println("  Name: " + rs.getString(3));
    System.out.println("  Address: " + rs.getString(4));
}

// Close the result set.
rs.close();

// Commit the select - yes selects need to be committed too
con.commit();

// Close the select statement - we're done with it
pSel.close();
```

Fetching multiple rows of data

Fetching multiple rows of data from a TimesTen data store can increase the performance of an application that connects to a data store set with Read committed isolation.

You can specify the number of rows to be prefetched by:

- Calling the **Statement.setFetchSize()** and **ResultSet.setFetchSize** methods. These are the standard JDBC calls, but the limitation is that they only affect one statement at a time.
- Calling the **TimesTenConnection.setTtPrefetchCount()** method or by using the **ttIsql** `prefetchcount` command. These enable a TimesTen extension that establishes prefetch on a connection level so that all of the statements on the connection use the same prefetch setting.

This section describes the connection-level prefetch implemented in TimesTen.

Note: You can only use the TimesTen prefetch count extension with direct-linked applications.

When the prefetch count is set to 0, TimesTen uses a default value, depending on the **Isolation** level you have set for the data store. In Read committed isolation mode, the default prefetch value is 128; in Read committed isolation mode, the default is 5. The default prefetch value is the optimum setting for most applications.

You can reset the prefetch value to any integer from 0 to 128, inclusive. When resetting the prefetch count, be aware that, when using Read committed isolation isolation, locks will be acquired until commit, so lock duration may increase if there is some delay between the fetch and execution. In Read committed isolation isolation, copies of the rows read will be retained until fetch so if those rows are deleted or updated, multiple copies may exist that consume space.

To disable prefetch, set the prefetch count to 1.

Call **TimesTenConnection.getTtPrefetchCount()** to check the current prefetch value.

Example 2.8 In this example, we use the **ttIsql** `prefetchcount` command to set the prefetch count for the connection to 6:

```
> ttIsql RunData_tt51
Command > prefetchcount 6;
```

Example 2.9 In this example, we use **setTtPrefetchCount()** to set the prefetch count to 10 and then use **getTtPrefetchCount()** to return the prefetch count in the count variable.

```
TimesTenConnection con =
     (TimesTenConnection) DriverManager.getConnection(url);

// set prefech count to 10 for this connection
con.setTtPrefetchCount(10);

// Return the prefetch count to the 'count' variable.
int count = con.getTtPrefetchCount();
```

Executing multiple SQL statements in a batch

You can improve performance by calling the **addBatch()** and **executeBatch()** methods for the **Statement** and **PreparedStatement** objects.

For **Statement** objects, a batch typically consists of a set of INSERT or UPDATE statements. Statements that return result sets are not allowed in a batch. A SQL statement is added to a batch by calling the **addBatch()** method. The set of SQL statements associated with a batch are executed through the **executeBatch()** method. For example:

```
// turn off autocommit
conn.setAutoCommit(false);

Statement stmt = conn.createStatement();
stmt.addBatch("INSERT INTO employees VALUES (1000, 'Joe
Jones')");
stmt.addBatch("INSERT INTO departments VALUES (260, 'Shoe')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1000, 260)");

// submit a batch of update commands for execution
int[] updateCounts = stmt.executeBatch();
conn.commit ();
```

For **PreparedStatement** objects, a batch consists of a set of prepared statement input parameters. Prepared statement parameters are added to the batch by executing **set** calls followed by the **addBatch()** call. The batch is executed via the **executeBatch()** method. For example:

```
// turn off autocommit
conn.setAutoCommit(false);

PreparedStatement stmt = conn.prepareStatement(
"INSERT INTO employees VALUES (?, ?)");

// first set of parameters
stmt.setInt(1, 2000);
stmt.setString(2, "Kelly Kaufmann");
stmt.addBatch();

// second set of parameters
stmt.setInt(1, 3000);
stmt.setString(2, "Bill Barnes");
stmt.addBatch();

// submit the batch for execution. Check update counts
int[] updateCounts = stmt.executeBatch();
conn.commit ();
```

Working with result sets

In addition to queries, some methods and built-in procedures return TimesTen data in the form of a **ResultSet** object. This section describes what you need to know when using **ResultSet** objects from TimesTen.

- TimesTen does not support multiple open **ResultSet** objects per statement. TimesTen cannot return multiple **ResultSet** objects from a single **Statement** object without first closing the current result set.
- TimesTen does not support holdable cursors. You cannot specify the holdability of a result set, i.e. whether a cursor can remain open after it has been committed.
- **ResultSet** objects are not scrollable or updatable, so you cannot specify **ResultSet.TYPE_SCROLL_SENSITIVE** or **ResultSet.CONCUR_UPDATABLE**.
- Use the **ResultSet.close** method to close **ResultSet** objects as soon as you are done with them.
- Calling **ResultSet.getString** is more costly performance-wise if the underlying data type is not a string. Because Java strings are immutable, **ResultSet.getString** must allocate space for a new string each time it is called. This makes **ResultSet.getString** one of the costlier calls in JDBC. Do not use **ResultSet.getString** to retrieve primitive numeric types, like Byte or Integer, unless it is absolutely necessary. For example, it is much faster to call **ResultSet.getInt** on an integer column.

Calling TimesTen built-in procedures

Chapter 3, "Built-In Procedures" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide* describes the TimesTen built-in procedures that extend standard ODBC functionality. You can execute a TimesTen built-in procedure using the **CallableStatement** interface.

To execute the built-in procedure, use the format:

```
CallableStatement.execute("{ Call Procedure }")
```

To prepare and execute a built-in procedure, use the format:

```
CallableStatement cStmt;
cStmt = con.prepareCall("{ Call Procedure }");
cStmt.execute();
```

For built-in procedures that return results, you can use the **ResultSet get*()** methods to retrieve the data from the returned **ResultSet**, as demonstrated in Example 2.11.

Note: See "Working with result sets" on page 42 for details on what you need to know when working with result sets generated by TimesTen.

Example 2.10 To call the **ttCkpt** procedure to initiate a fuzzy checkpoint, enter:

```
Connection con;
CallableStatement cStmt;
.......
cStmt = con.prepareCall("{ Call ttCkpt }");
cStmt.execute();
con.commit();          // commit the transaction
```

Example 2.11 This example calls the **ttDataStoreStatus** procedure and prints out the returned result set.

Contrary to the advice given in "Working with result sets" on page 42, we use **ResultSet.getString** in this example to retrieve the *Context* field, which is a binary. This is because the data is output is printed, rather than used for processing. If you were not to print the *Context* value, you could achieve better performance using the **ResultSet.getBytes** method.

```
ResultSet rs;

cStmt = con.prepareCall("{ Call ttDataStoreStatus }");

if (cStmt.execute() == true) {
     rs = cStmt.getResultSet();
     System.out.println("Fetching result set...");
     while (rs.next()) {
        System.out.println("\n Data store: " + rs.getString(1));
        System.out.println("  PID: " + rs.getInt(2));
        System.out.println("  Context: " + rs.getString(3));
        System.out.println("  ConType: " + rs.getString(4));
        System.out.println("  memoryID: " + rs.getString(5));
        }
    rs.close();
  }
cStmt.close();
```

Note: You cannot pass parameters to **CallableStatement** by name. You must set parameters by ordinal numbers. And you cannot use the SQL escape syntax.

Managing Multiple Threads

Note: On some UNIX platforms, it is necessary to set the THREADS_FLAG, as described in "Set the THREADS_FLAG variable (UNIX only)" on page 15.

The `level4.java` demo demonstrates the use of multiple threads.

When your application has a direct driver connection to the data store, TimesTen functions share stack space with your application. In multi-threaded environments, it is important to avoid overrunning the stack allocated to each thread because consequences can result that are unpredictable and difficult to debug. The amount of stack space consumed by TimesTen calls varies depending on the SQL functionality used. Most applications should set their thread stack space to at least 16 KB on 32-bit systems and between 34 KB to 72 KB on 64-bit systems.

The amount of stack space allocated per thread is specified by the operating system when threads are created. On Windows, you can use the TimesTen debug driver and link your application against the Visual C++ debug C library to enable "stack probes" that raise an identifiable exception if a thread attempts to grow its stack beyond the amount allocated.

Note: In multi-threaded applications, a thread that issues requests on different connection handles to the same data store may encounter lock conflict with itself. TimesTen resolves these conflicts with lock timeouts.

Handling Errors

This section discusses how to check for, identify and handle errors in you TimesTen Java application.

For a list of the errors that TimesTen returns and what to do if the error is encountered, see Chapter 6, "Warnings and Errors" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.

The topics are:

- About fatal errors, non-fatal errors, and warnings
- Reporting errors and warnings
- Detecting and responding to specific errors
- Rolling back failed transactions

About fatal errors, non-fatal errors, and warnings

TimesTen can return a fatal error, a non-fatal error, or a warning.

Handling fatal errors and recovery

Fatal errors are those that make the data store inaccessible until it can be recovered. When a fatal error occurs, all data store connections are required to disconnect. No further operations may complete. Fatal errors are indicated by TimesTen error codes 846 and 994. Error handling for these errors should be different from standard error handling. In particular, the code should rollback the transaction and disconnect from the data store.

When fatal errors occur, TimesTen performs the full cleanup and recovery procedure:

- Every connection to the data store is invalidated, a new memory segment is allocated and applications are required to disconnect.
- The data store is recovered from the checkpoint and log files upon the first subsequent initial connection.
  - The recovered data store reflects the state of all durably committed transactions and possibly some transactions that were committed non-durably.
  - No uncommitted or rolled back transactions are reflected.

If no checkpoint or log files exist and **AutoCreate** is set, TimesTen creates an empty data store.

Handling non-fatal errors

Non-fatal errors include simple errors such as an INSERT that violates unique constraints. This category also includes some classes of application and process failures.

TimesTen returns non-fatal errors through the normal error-handling process and requires the application to check for and identify them.

When a data store is affected by a non-fatal error, an error may be returned and the application should take appropriate action. In some cases, such as in the case of a process crash, an error cannot be returned, so TimesTen automatically rolls back the failed process' transactions.

An application can handle non-fatal errors by modifying its actions or, in some cases, by rolling back one or more offending transactions, as described in "Rolling back failed transactions" on page 51.

Note: If a **ResultSet**, **Statement**, **PreparedStatement**, **CallableStatement** or **Connection** operation results in a data store error, it is a good practice to call the **close** method for that object.

About warnings

TimesTen returns warnings when something unexpected occurs that you may want to know about. Some examples of events that cause TimesTen to issue a warning include:

- A checkpoint failure
- The use of a deprecated TimesTen feature
- The truncation of some data
- The execution of a recovery process upon connect

You should always include code that checks for warnings, as they can indicate application problems.

Reporting errors and warnings

You should check for and report all errors and warnings that can be returned on every call. This saves considerable time and effort during development and debugging. A **SQLException** object is generated in case of one or more data store access errors and a **SQLWarning** object is generated in the case of one or more warning messages. A single call may return multiple errors and/or warnings, so your application should report all errors or warnings in the returned **SQLException** or **SQLWarning** objects.

Multiple errors or warnings are returned in linked chains of **SQLException** or **SQLWarning** objects. Example 2.12 and Example 2.13 demonstrate how you might iterate through the lists of returned **SQLException** and **SQLWarning** objects to report all of the errors and warnings, respectively.

Example 2.12 This method prints out the content of all exceptions in the linked **SQLException** objects.

```
static int reportSQLExceptions(SQLException ex)
  {
    int errCount = 0;

    if (ex != null) {
      errStream.println("\n--- SQLException caught ---");
      ex.printStackTrace();

      while (ex != null) {
        errStream.println("SQL State:   " + ex.getSQLState());
        errStream.println("Message:     " + ex.getMessage());
        errStream.println("Error Code: " + ex.getErrorCode());
        errCount ++;
        ex = ex.getNextException();
        errStream.println();
      }
    }

    return errCount;
  }
```

Example 2.13 This method prints out the content of all warning in the linked **SQLWarning** objects.

```
static int reportSQLWarnings(SQLWarning wn)
{
   int warnCount = 0;

   while (wn != null) {
      errStream.println("\n--- SQL Warning ---");
      errStream.println("SQL State:   " + wn.getSQLState());
      errStream.println("Message:     " + wn.getMessage());
      errStream.println("Error Code: " + wn.getErrorCode());

      // is this a SQLWarning object or a DataTruncation object?
      if (wn instanceof DataTruncation) {
         DataTruncation trn = (DataTruncation) wn;
         errStream.println("Truncation error in column: " +
            trn.getIndex());
      }

      warnCount++;
      wn = wn.getNextWarning();
      errStream.println();
   }

   return warnCount;
}
```

Detecting and responding to specific errors

In some situations it may be desirable to respond to a specific SQL state or TimesTen error code. You can use **SQLException.getSQLState** to return the SQL99 SQL state error string, and **SQLException.getErrorCode** to return TimesTen error codes, as shown in Example 2.14.

Example 2.14 The TimesTen demos require that you load the demo schema before they are executed. The following **catch** statement alerts the user that the demo schema has not been loaded or has not been refreshed by detecting ODBC error S0002 and TimesTen error 907:

```
catch (SQLException ex) {
  if (ex.getSQLState().equalsIgnoreCase("S0002")) {
     errStream.println("\nError: The table xyz.customer
         does not exist.\n\t Please run ttIsql -f input0.dat
         to initialize the database.");
  } else if (ex.getErrorCode() == 907) {
     errStream.println("\nError: Attempting to insert a row
         with a duplicate primary key.\n\tPlease rerun ttIsql -f
         input0.dat to reinitialize the database.");
  }
```

You can use the **TimesTenVendorCode** interface to detect the errors by their name, rather than their number.

For example:

```
ex.getErrorCode() ==
com.timesten.jdbc.TimesTenVendorCode.TT_ERR_KEYEXISTS
```

is the same as:

```
ex.getErrorCode() == 907
```

See "TimesTenVendorCode" on page 103 for the complete list of error name-to-number mappings.

Rolling back failed transactions

In some situations, such as to recover from a deadlock or time-out condition, you may want to explicitly roll back the transaction using the **Connection.rollback()** method.

For example:

```
try {
  if (con != null && !con.isClosed()) {
   // Rollback any transactions in case of errors
     if (retcode != 0) {
       try {
         System.out.println("\nEncountered error.
                                Rolling back transaction");
         con.rollback();
       } catch (SQLException ex) {
           reportSQLExceptions(ex);
         }
      }

  System.out.println("\nClosing the connection\n");
  con.close();
}
```

The XACT_ROLLBACKS column of the SYS.MONITOR table indicates the number of transactions that were rolled back.

A transaction rollback consumes resources and the entire transaction is in effect wasted. To avoid unnecessary rollbacks, design your application to avoid contention (see "Choose the best method of locking" on page 79) and check application or input data for potential errors before submitting it, whenever possible.

Note: Should your application crash in the middle of an active transaction, TimesTen automatically rolls back the transaction.

3

Writing an XLA Event Handler

You can use the TimesTen JMS/XLA API (JMS/XLA) to monitor TimesTen for changes to specified tables in a local data store and receive real-time notification of these changes. JMS/XLA implements Sun Microsystems' Java Message Service (JMS) interfaces to make the functionality of the TimesTen Transaction Log API (XLA) available to Java applications.

For an introduction to TimesTen event management, see "Event Management" in the *TimesTen Architectural Overview*. For detailed information about the JMS API, refer to the JMS 1.1 documentation published by Sun Microsystems.[1]

For information on tuning TimesTen JMS/XLA applications for improved performance, see "Tuning JMS/XLA applications" on page 86.

JMS/XLA Concepts

Java applications can use the JMS/XLA API to receive event notifications from TimesTen. JMS/XLA uses the JMS publish-subscribe interface to provide access to XLA updates.

You subscribe to updates by establishing a JMS **Session** that provides a connection to XLA and creating a durable subscriber (**TopicSubscriber**). You can receive and process messages synchronously through the subscriber, or you can implement a listener (**MessageListener**) to process the updates asynchronously.

JMS/XLA is designed for applications that want to monitor a local datastore--TimesTen and the application receiving the notifications must reside on the same machine.

Note: The JMS/XLA API supports persistent-mode XLA. In this mode, XLA obtains update records directly from the transaction log buffer or log files, so the records are available until they are read. Persistent-mode XLA also allows multiple readers to access transaction log updates simultaneously.

---

1. The JMS javadoc is installed with the TimesTen In-Memory Database. For additional information about JMS, go to http://java.sun.com/products/jms/docs.html.

How XLA reads records from the transaction log

As applications modify a TimesTen data store, TimesTen generates log records that describe the changes made to the data and other events such as transaction commits.

New log records are always written to the end of the log buffer as they are generated. When disk-based logging is enabled for the data store, log records are periodically flushed in batches from the log buffer in memory to log files on disk.

Applications can use XLA to monitor the transaction log for changes to the TimesTen data store. XLA reads through the transaction log, filters the log records, and delivers XLA applications with a list of transaction records that contain the changes to the tables and columns of interest.

XLA sorts the records into discrete transactions. If multiple applications are updating the data store simultaneously, log records from the different applications will be interleaved in the log.

XLA transparently extracts all log records associated with a particular transaction and delivers them in a contiguous list to the application.

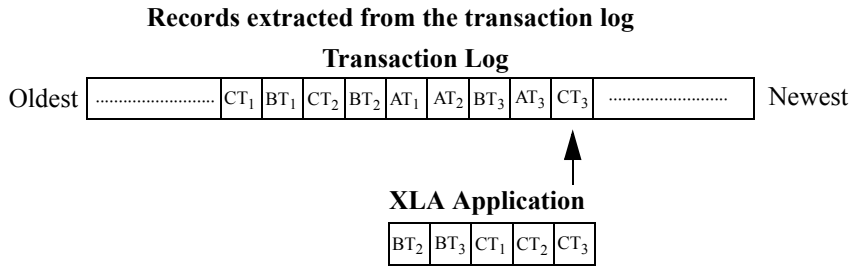
Only the records for committed transactions are returned. They are returned in the order in which their final commit record appears in the transaction log. XLA filters out records associated with changes to the data store that have not yet committed.

If a change is made but then rolled back, XLA does not deliver the records for the aborted transaction to the application.

Most of these basic XLA concepts are demonstrated in Example 3.1 and summarized in the bulleted list on the bottom.

Example 3.1 Consider the example transaction log illustrated in Figure 3.1.

Figure 3.1



In this example, the transaction log contains the following records:

```
CT1 - Application C updates row 1 of table W with value 7.7
BT1 - Application B updates row 3 of table X with value 2
```

CT_2 - Application C updates row 9 of table W with value 5.6

BT_2 - Application B updates row 2 of table Y with value XYZ

AT_1 - Application A updates row 1 of table Z with value 3

AT_2 - Application A updates row 3 of table Z with value 4

BT_3 - Application B commits its transaction

AT_3 - Application A rolls back its transaction

CT_3 - Application C commits its transaction

An XLA application that is set up to detect changes to *Tables W, Y,* and *Z* would see:

BT_2 and BT_3 - Update row 2 of table Y with value XYZ and commit

CT_1 - Update row 1 of table W with value 7.7

CT_2 and CT_3 - Update row 9 of table W with value 5.6 and commit

What this example demonstrates:

- *Application B* and *C's* transaction records all appear together.
- Though the records for *Application C* begin to appear in the transaction log before those for *Application B*, the commit for *Application B* (BT_3) appears in the log before the commit for *Application C* (CT_3). As a result, the records for *Application B* are returned to the XLA application ahead of those for *Application C*.
- *Application B's* update to *Table X* (BT_1) aren't presented because XLA is not set up to detect changes to *Table X*.
- *Application A's* updates to *Table Z* (AT_1 and AT_2) are never presented, since it didn't commit and was rolled back (AT_3).

XLA and materialized views

You can use XLA to track changes to both tables and materialized views. A materialized view provides a single source from which you can track changes to selected rows and columns in multiple *detail* tables. Without a materialized view, the XLA application would have to monitor and filter the update records from all of the detail tables, including records reflecting updates to rows and columns of no interest to the application.

In general, there are no operational differences between the XLA mechanisms used to track changes to a table or a materialized view.

XLA topics

To connect to XLA, you establish a connection to a JMS `Topic` that corresponds to a particular datastore. A configuration file, `jmsxla.xml`, provides the mapping between topic names and datastores.

A topic definition in `jmsxla.xml` consists of a name, a JDBC connection string, and a prefetch value that specifies how many updates to retrieve at a time. You can also include the `enabledTables` attribute to automatically enable XLA publishing for particular tables when the application connects to the topic. (This is useful for applications that monitor a static set of tables. For information about enabling XLA publishing dynamically, see "Monitoring Tables for Updates" on page 60.)

For example, the configuration file shown in Example 3.2 maps the `DemoDataStore` topic to the `TestDB` DSN and enables update notifications for the `xyz.customer` table.

Example 3.2

```
<xlaconfig>
  <topics>
    <topic name="DemoDataStore"
      connectionString="jdbc:timesten:direct:DSN=TestDB"
      xlaPrefetch="100"
      enabledTables="xyz.customer"
    />
  </topics>
</xlaconfig>
```

XLA updates

Applications receive XLA updates as JMS **MapMessage** objects. The `MapMessage` contains a set of typed name/value pairs that correspond to the fields in an XLA update header.

You can access the message fields using the `MapMessage` *`get`* methods. The `getMapNames` method returns an `Enumeration` that contains the names of all of the fields in the message. You can retrieve individual fields from the message by name. All reserved field names begin with two underscores, for example `__TYPE`.

All update messages have a `__TYPE` field that indicates what type of update the message contains. The types are specified as integer values. As a convenience, you can use the constants defined in `com.timesten.dataserver.jmsxla.XlaConstants` to compare against the integer types. The supported types are described in Table 3.1.

Table 3.1 XLA Update Types

Update Type	Description
`INSERT`	A row has been added.
`UPDATE`	A row has been modified.
`DELETE`	A row has been removed.

Table 3.1 XLA Update Types

Update Type	Description
COMMIT_ONLY	A transaction has been committed.
CREATE_TABLE	A table has been created.
DROP_TABLE	A table has been dropped.
CREATE_INDEX	An index has been created.
DROP_INDEX	An index has been dropped.
ADD_COLUMNS	New columns have been added to the table.
DROP_COLUMNS	Columns have been removed from the table.
CREATE_VIEW	A materialized view has been created.
DROP_VIEW	A materialized view has been dropped.
CREATE_SEQ	A SEQUENCE has been created.
DROP_SEQ	A SEQUENCE has been dropped.
TRUNCATE	The table has been truncated and all rows in the table have been deleted.

For more information about the contents of an XLA update message see "XLA MapMessage contents" on page 159.

XLA bookmarks

An XLA bookmark marks an XLA subscriber application's read position in the transaction log. Bookmarks facilitate durable subscriptions, enabling an application to disconnect from a topic, and then reconnect to continue receiving updates where it left off.

When you create a message consumer for XLA, you always use a durable **TopicSubscriber**. The subscription identifier you specify when you create the subscriber is used as the XLA bookmark name. When you use the built-in procedures `ttXlaSubscribe` and `ttXlaUnsubscribe` via JDBC to start and stop XLA publishing for a table, you explicitly specify the name of the bookmark to be used.

Bookmarks are reset to the last read position whenever an acknowledgement is received. For more information about how update messages are acknowledged, see "XLA acknowledgement modes" on page 58.

You can remove a durable subscription by calling `unsubscribe` on your JMS **`Session`**. This deletes the corresponding XLA bookmark and forces a new subscription to be created when you reconnect. For more information see "Deleting bookmarks" on page 64.

XLA acknowledgement modes

XLA's acknowledgement mechanism is designed to ensure that an application hasn't just received a message, but has successfully processed it. Acknowledging an update permanently resets the application's XLA bookmark to the last record that was read. This prevents previously returned records from being re-read, ensuring that an application does not receive previously acknowledged records if the bookmark is reused when an application reconnects to XLA.

JMS/XLA can automatically acknowledge XLA update messages, or applications can choose to acknowledge messages explicitly. You specify how updates are to be acknowledged when you create the `Session`.

JMS/XLA supports three acknowledgement modes:

- AUTO_ACKNOWLEDGE—In this mode, updates are automatically acknowledged as you receive them. Each message is delivered only once--duplicate messages will not be sent and in the event of an application failure, messages might be lost. In AUTO_ACKNOWLEDGE mode, messages are always delivered and acknowledged individually, so JMS/XLA does not prefetch multiple records. (The `xlaprefetch` attribute in the topic is ignored.)
- DUPS_OK_ACKNOWLEDGE—In this mode, updates are automatically acknowledged, but duplicate messages might be delivered in the event of an application failure. In DUPS_OK_ACKNOWLEDGE mode, JMS/XLA prefetches records according to the `xlaprefetch` attribute specified for the topic and sends an acknowledgement when the last record in a prefetched block is read. If the application fails before reading all of the prefetched records, the all of the records in the block are presented to the application it restarts.
- CLIENT_ACKNOWLEDGE—in this mode, applications are responsible for acknowledging receipt of update messages by calling `acknowledge` on the `MapMessage`. In CLIENT_ACKNOWLEDGE mode, JMS/XLA prefetches records according to the `xlaprefetch` attribute specified for the topic.

Prefetching updates

Prefetching multiple update records at a time is more efficient than obtaining each update record from XLA individually. Because updates are not prefetched when you use AUTO_ACKNOWLEDGE mode, it can be slower than the other modes. If possible, you should design your application to tolerate duplicate updates so you can use DUPS_OK_ACKNOWLEDGE, or explicitly acknowledge updates. Explicitly acknowledging updates usually yields the best

performance, as long as you can avoid acknowledging each message individually.

Acknowledging updates

To explicitly acknowledge an XLA update, you call `acknowledge` on the update message. Acknowledging a message implicitly acknowledges all previous messages. Typically, you receive and process multiple update messages between acknowledgements. If you are using the CLIENT_ACKNOWLEDGE mode and intend to reuse a durable subscription in the future, you should call `acknowledge` to reset the bookmark to the last-read position before exiting.

XLA Demos

The TimesTen JMS/XLA demos demonstrate how to use the JMS/XLA API to subscribe to updates to a TimesTen table. The demo applications are located in the *`install_dir`*`/demo/tutorial/java` directory.

The demos use the TimesTen demo schema, which simulates a simple order-processing database. For information about compiling the TimesTen Java demos and building the demo schema, see "About the TimesTen Java Demos."

XlaLevel1 demo

The procedures demonstrated in this section are based on the `XlaLevel1.java` demo application. This application reports on updates to the `xyz.customer` table.

Connecting to XLA

To connect to XLA so you can receive updates, you use the JMS **ConnectionFactory** to create a `Connection`. You then use the `Connection` to establish a **Session**. When you're ready to start processing updates, you call `start` on the **Connection** to enable message dispatching.

For example, in the XlaLevel1 demo, the JMS `Session` is set up by the `XLALevel1 subscribe` method, as shown in Example 3.3. Note that the acknowledgement mode is set when the session is created.

Example 3.3

```
ConnectionFactory connectionFactory;

Context messaging = new InitialContext();
connectionFactory = (ConnectionFactory)
      messaging.lookup("ConnectionFactory");
Connection connection = connectionFactory.createConnection();

Session session = connection.createSession(false,
      Session.AUTO_ACKNOWLEDGE);
```

```
connection.start();
```

Monitoring Tables for Updates

Before you can start receiving updates, you need to tell XLA which tables you want to monitor for changes. You can either specify the tables in the `jmsxla.xml` file, or call built-in procedures via JDBC to explicitly enable and disable table monitoring.

To subscribe to changes and turn on XLA publishing for a table, you either specify the `enabledTables` attribute in the `jmsxla.xml` file, or call the `ttXlaSubscribe` built-in procedure via JDBC. Including the `enabledTables` attribute in the configuration file provides a simple way to subscribe to changes if your application monitors a static set of tables.

When you use `ttXlaSubscribe` to enable XLA publishing for a table, you need to specify two parameters, the name of the table and the name of the bookmark that will be used to track the table:

```
ttXlaSubscribe(user.table, bookmark)
```

For example, to call `ttXlaSubscribe` via the JDBC `CallableStatement` interface:

```
Connection con;
CallableStatement cStmt;
...
cStmt = con.prepareCall("{ttXlaSubscribe(user.table, bookmark)}");
cStmt.execute();
```

If you subscribe to table updates by calling `ttXlaSubscribe`, you should use `ttXlaUnsubscribe` to unsubscribe from the table during shut down. For more information, see "Unsubscribing from a table" on page 64.

For more information about using TimesTen built-in procedures in a Java application, see "Calling TimesTen built-in procedures" in Chapter 2 of the TimesTen Java Developer's and Reference Guide. See the TimesTen API and SQL Reference Guide for more information about the built-in procedures themselves.

Receiving and Processing Updates

You can receive XLA updates either synchronously or asynchronously. To receive and process update for a topic synchronously, you:

1. Create a durable **TopicSubscriber** to subscribe to a topic.
2. Call `receive` or `receiveNoWait` on your subscriber to get the next available update.
3. Process the returned **MapMessage**.

To receive and process updates for a topic asynchronously, you:

1. Create a **MessageListener** that will process the updates.
2. Create a durable `TopicSubscriber` to subscribe to a topic.
3. Register your `MessageListener` with the `TopicSubscriber`.
4. Wait for messages to arrive. You can call `Object.wait` to wait for messages if your application doesn't need to do anything else in its main thread.

   When an update is published, your `MessageListener onMessage` method is called and the message is passed in as a `MapMessage`.

   For example, the XlaLevel1 demo uses a listener to process updates asynchronously, as shown in Example 3.4.

Example 3.4

```
MyListener myListener = new MyListener(outStream);

outStream.println("Creating consumer for topic " + topic);
Topic xlaTopic = session.createTopic(topic);
```

```
TopicSubscriber subscriber =
      session.createDurableSubscriber(xlaTopic, "mybookmark");

subscriber.setMessageListener(myListener);
```

Note that the `TopicSubscriber` must be a durable subscriber. XLA connections are designed to be durable—XLA bookmarks make it possible to disconnect from a topic, and then reconnect to start receiving updates where you left off. The `String` you pass in as the subscription identifier when you create a durable subscriber is used as the XLA bookmark name.

You can call `unsubscribe` on the JMS **TopicSession** when your application shuts down to delete the XLA bookmark used by the subscriber. This causes a new bookmark to be created when the application is restarted.

Processing updates

When you receive an update, you can use the **MapMessage** *`get`* methods to extract information from the message and then perform whatever processing your application requires. The TimesTen `XlaConstants.java` class defines constants for the update types and special message fields to make it easier to process XLA update messages.

The first step is typically to determine what type of update the message contains. You can use the `MapMessage.getInt` method to get the contents of the `__TYPE` field, and compare the value against the numeric constants defined in `the` XlaConstants class.

In the `XlaLevel1` demo, MySubscriber's `onMessage` method extracts the update type from the `MapMessage` and displays the action that the update signifies. This is shown in Example 3.5.

Example 3.5

```
public void onMessage(Message message) {
      MapMessage mapMessage = (MapMessage) message;
      String messageType = null;
      if (message == null) {
        errStream.println("MyListener: update message is null");
        return;
      }

      try {
        // Get the update type(insert, update, delete, etc.).
        int type = mapMessage.getInt(XlaConstants.TYPE_FIELD);
        if (type == XlaConstants.INSERT) {
          System.out.println("A row was inserted.");
        } else if (type == XlaConstants.UPDATE) {
          System.out.println("A row was updated.");
        } else if (type == XlaConstants.DELETE) {
```

```
            System.out.println("A row was deleted.");
          } else {
            return;
          }
```

Once you know what type of message you have received, you can process the message according to your application's needs. To get a list of all of the fields in a message, you can call `MapMessage.getMapNames`. You can retrieve individual fields from the message by name.

For example, the XlaLevel1 demo extracts the column values from insert, update, and delete messages using the column names, as shown in Example 3.6.

Example 3.6

```
if (type == XlaConstants.INSERT
       || type == XlaConstants.UPDATE
       || type == XlaConstants.DELETE) {
          // Get the column values from the message.
          int cust_num = mapMessage.getInt("cust_num");
          String region = mapMessage.getString("region");
          String name = mapMessage.getString("name");
          String address = mapMessage.getString("address");

          System.out.println("New Column Values:");
          System.out.println("cust_num=" + cust_num);
          System.out.println("region=" + region);
          System.out.println("name=" + name);
          System.out.println("address=" + address);
        }
```

For detailed information about the contents of XLA update messages, see "XLA MapMessage contents" on page 159. For information about how TimesTen column types map to JMS data types and the *get* methods used to retrieve the column values, see "Data type mapping" on page 171.

Terminating an XLA Application

When your XLA application has finished reading from the transaction log, you should gracefully exit by closing the XLA Connection, deleting any unneeded bookmarks, and unsubscribing from any tables to which you explicitly subscribed.

Closing the connection

To close the connection to XLA, you call `close` on the **Connection** object.

Once a connection has been closed, any attempt to use it, its sessions, or its subscribers will throw an `IllegalStateException`. You can continue to use

messages received via the connection, but you cannot call a received message's `acknowledge` method once the connection is closed.

Deleting bookmarks

Deleting XLA bookmarks during shutdown is optional. Deleting a bookmark enables the disk space associated with any unread update records in the transaction log to be freed.

If you *do not* delete the bookmark, it can be reused by a durable subscriber. As long as the bookmark is available when a durable subscriber reconnects, the subscriber will receive all unacknowledged updates published since the previous connection was terminated. Keep in mind that as long as a bookmark exists with no application reading from it, the transaction log will continue to grow and the amount of disk space consumed by your database will increase.

To delete a bookmark, you can simply call `unsubscribe` on the JMS Session, which invokes the `ttXlaDeleteBookmark` built-in procedure to remove the XLA bookmark.

Unsubscribing from a table

To turn off XLA publishing for a table, you use the `ttXlaUnsubscribe` built-in procedure. If you use `ttXlaSubscribe` to enable XLA publishing for a table, you should use `ttXlaUnsubscribe` to unsubscribe from the table when shutting down your application.

When you unsubscribe from a table, you specify the name of the table and the name of the bookmark used to track the table:

```
ttXlaUnsubscribe(user.table, bookmark)
```

For example, to call `ttXlaSubscribe` via the JDBC `CallableStatement` interface:

```
Connection con;
CallableStatement cStmt;
...
cStmt = con.prepareCall("{ttXlaUnSubscribe(user.table, bookmark)}");
cStmt.execute();
```

For more information about using TimesTen built-in procedures in a Java application, see "Calling TimesTen built-in procedures" in Chapter 2 of the TimesTen Java Developer's and Reference Guide. See the TimesTen API and SQL Reference Guide for more information about the built-in procedures themselves.

4

Distributed Transaction Processing: JTA

This chapter describes the TimesTen implementation of the Java Transaction API (JTA).

The TimesTen implementation of the Java JTA interfaces is intended to enable Java applications, application servers, and transaction managers to use TimesTen resource managers in DTP environments. The TimesTen implementation is supported for use by the BEA WebLogic application server.

The purpose of this chapter is to provide information specific to the TimesTen implementation of XA and JTA and is intended to be used with the following documents:

- The Java Transaction API (JTA) and JDBC documentation available from the Sun Microsystems web site:

  ```
  http://java.sun.com/products/jta/
  http://java.sun.com/products/jdbc/
  ```

- BEA documentation available from the BEA Systems web site:

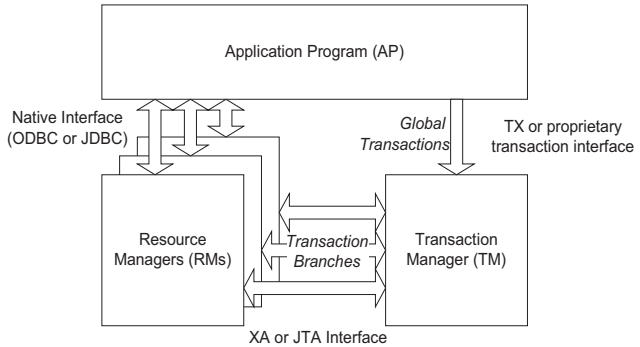
  ```
  http://edocs.bea.com/
  ```

The main topics in this chapter are:

X/Open DTP Model

Figure 4.1 illustrates the interfaces defined by the X/Open DTP model.

Figure 4.1 Distributed transaction processing model



In the DTP model, a *transaction manager* breaks each *global transaction* down into multiple *branches* and distributes them to separate *resource managers* for service. In the context of TimesTen XA, the resource managers can be a collection of TimesTen data stores, or data stores in combination with other commercial databases that support XA.

As shown in Figure 4.1, applications use the *TX* interface to communicate global transactions to the transaction manager. The transaction manager breaks the global transaction down into branches and uses the *XA* interface to coordinate each transaction branch with the appropriate resource manager.

Global transaction control provided by the TX and XA interfaces is distinct from local transaction control provided by the native ODBC and JDBC interfaces. It is generally best to maintain separate connections for local and global transactions. Applications can obtain a connection handle to a TimesTen resource manager in order to initiate both local and global transactions over the same connection.

Two-phase commit

In an XA implementation, the transaction manager commits the distributed branches of a global transaction by means of a *two-phase commit* protocol. The two phases are:

- In *phase one* of the commit, the transaction manager directs each resource manager to *prepare to commit*, which is to verify and guarantee it can commit its respective branch of the global transaction. If a resource manager is unable to commit its branch, the transaction manager *rolls back* the entire transaction in phase two.
- In *phase two* of the commit, the transaction manager either directs each resource manager to commit its branch or, if a transaction manager reported it was unable to commit in phase one, rolls back the global transaction.

See the optimizations described below for exceptions.

Note: The transaction manager considers the global transaction committed if and only if all branches successfully commit.

One-phase commit optimization

If a global transaction is determined by the transaction manager to have involved only one branch, it skips phase one and commits the transaction in phase two.

Read-only optimization

If a global transaction branch is read-only, the transaction manager commits the branch in phase one and skips phase two. A "read-only" transaction is one that does not generate any log records.

Setting TimesTen data store attributes for XA

This section describes how to set your TimesTen data store attributes for TimesTen XA.

DurableCommits

In order to guarantee global transaction consistency, TimesTen XA transaction branches must be durable. The TimesTen implementation of the **xa_prepare()**, **xa_rollback()**, and **xa_commit()** functions log their actions to disk, regardless of the value set in the **DurableCommits** connection attribute or by the **ttDurableCommit** built-in procedure. Should you need to recover from a failure, both the resource manager and the TimesTen transaction manager will have a consistent view of which transaction branches were active (in a prepared state) at the time of failure.

See "Recovering Global Transactions" on page 68 for more information.

Logging

In order to allow for rollback of transactions, you must have disk-based logging enabled (**Logging**=1).

Operation with diskless or no logging is not allowed with XA.

Logging=0 with XA results in the error:

```
Error 11033 - "XA support requires that logging be enabled"
```

Logging=2 with XA results in the error:

```
Error 11033 "XA support requires that logging to disk be enabled"
```

Recovering Global Transactions

When a TimesTen data store is loaded from disk after a crash or unexpected termination, and recovery takes place, any global transactions that were prepared but not committed are left pending, or *in-doubt*. Normal processing is not enabled until the disposition of all in-doubt transactions has been resolved. The resolution procedure is accomplished as follows:

Once connect and recovery are complete, TimesTen checks for in-doubt transactions. If there are no in-doubt transactions, operation proceeds as normal. If there are in-doubt transactions, other connections may be created, but virtually all operations are prohibited on those connections until the in-doubt transactions are resolved. Any other ODBC or JDBC calls result in the error:

```
Error 11035 - "In-doubt transactions awaiting resolution in recovery
must be resolved first"
```

The list of in-doubt transactions can be retrieved through the XA or JTA implementation of **xa_recover()**, and then dealt with through XA or JTA **xa_commit()**, **xa_rollback()**, or **xa_forget()**, as appropriate. Once all the in-doubt transactions are cleared, operation proceeds normally.

This scheme should be adequate for systems that operate strictly under control of the transaction manager, since the first thing the transaction manager should do after connect is to call **xa_recover()**.

Note: TimesTen does not allow you to replicate TimesTen resource managers, as failovers to subscribers would result in inconsistent global transactions.

Heuristic recovery

If the transaction manager is unavailable or is unable to resolve an in-doubt transaction, you can use the **ttXactAdmin** utility to independently commit or abort the individual transaction branches. See the section on "ttXactAdmin" in the *TimesTen Reference Guide* for more information.

Note: The XID defined by XA standard has some of its members (such as, `formatID`, `gtrid_length`, and `bqual_length`) defined as type 'long'. This causes problem when 32-bit client applications connect to a 64-bit server or vice-versa because 'long' is a 32-bit integer on 32-bit platforms and 64-bit integer on 64-bit platforms (except 64-bit Windows). Hence, TimesTen internally uses only the 32 least significant bits of those XID members irrespective of the platform type of client or server. TimesTen does not support any value in those XID members that does not fit in a 32-bit integer.

Using the JTA API

The TimesTen implementation of JTA provides an API consistent with that specified by Sun Microsystems in the *Java Transaction API (JTA)* specification. TimesTen JTA operates on JDK 1.4 and above.

This section describes what you need to know when using TimesTen JTA.

What you need to know	Where to find out
How to set the TimesTen data store attributes when using XA	"Setting TimesTen data store attributes for XA" on page 67
What Java packages to import	"Required Packages" on page 70
How to create a TimesTen **XAConnection** object	"Creating a TimesTen XAConnection object" on page 70
How to recover in-doubt global transactions	"Recovering Global Transactions" on page 68
How to register a TimesTen DSN with BEA WebLogic	"Registering a TimesTen XADataSource with the BEA WebLogic server" on page 74
Error messages specific to TimesTen XA	"Error Handling" on page 75

Primary Documents

This section provides the TimesTen information to be used with the following documents:

The BEA *Programming WebLogic JTA* document:

```
http://e-docs.bea.com/wls/docs61/jta/index.html
```

The Sun Microsystems JTA reference pages:

```
http://java.sun.com/products/jta/javadocs-1.0.1/index.html
```

The Sun Microsystems JDBC reference pages:

```
http://java.sun.com/products/jdbc/jdbc20.stdext.javadoc/
```

Required Packages

The TimesTen JDBC and XA packages are available in:

```
com.timesten.jdbc.*;
com.timesten.jdbc.xa.*;
```

Your application should import the following packages:

```
import java.sql.*;
import javax.sql.*;
import javax.transaction.xa.*;
```

Creating a TimesTen XAConnection object

Connections to XA data sources are established through **XADataSource** objects. You can create a new **XAConnection** object for your TimesTen data source by using the **TimesTenXADataSource** object as a connection factory. **TimesTenXADataSource** implements the **javax.sql.XADataSource** interface:

```
http://java.sun.com/j2se/1.4.2/docs/api/javax/sql/
XADataSource.html
```

After creating a new **TimesTenXADataSource** instance, use the **setUrl()** method to specify a TimesTen data store. The URL should look like:

For a direct connection:

```
jdbc:timesten:direct:DSNname
```

For a client connection:

```
jdbc:timesten:client:DSNname
```

You can also optionally use the **setUser()** and **setPassword()** methods to set the id and password for a specific user.

Example 4.1 In this example, we use the **TimesTenXADataSource** object as a factory to create a new TimesTen XA data source object. We then set the URL that identifies the TimesTen DSN (*dsn1*), the user name (*myName*), and the password (*myPasswd*) for this **TimesTenXADataSource** instance.

We then use the **getXAConnection()** method to return a connection to the object, called *xaConn*.

```
TimesTenXADataSource xads = new TimesTenXADataSource();

xads.setUrl("jdbc:timesten:direct:dsn1");
xads.setUser("myName");
xads.setPassword("myPassword");

XAConnection xaConn = null;
try {
    xaConn = xads.getXAConnection();
}
catch (SQLException e){
    e.printStackTrace();
    return;
}
```

You can create multiple connections to an XA data source object. In Example 4.1, you could create second connection, called *xaConn2*, with:

```
XAConnection xaConn  = null;
XAConnection xaConn2 = null;
try {
    xaConn  = xads.getXAConnection();
    xaConn2 = xads.getXAConnection();
}
```

Example 4.2 In this example, we create two instances of **TimesTenXADataSource** for the TimesTen data sources named *dsn1* and *dsn2*. We then create a connection for *dsn1* and two connections for *dsn2*. All connections are to the same resource manager.

```
TimesTenXADataSource xads = new TimesTenXADataSource();

xads.setUrl("jdbc:timesten:direct:dsn1");
xads.setUser("myName");
xads.setPassword("myPassword");

XAConnection xaConn1 = null;
XAConnection xaConn2 = null;
XAConnection xaConn3 = null;

try {
   xaConn1 = xads.getXAConnection(); // connect to dsn1
}
catch (SQLException e){
   e.printStackTrace();
   return;
}

xads.setUrl("jdbc:timesten:direct:dsn2");
xads.setUser("myName");
xads.setPassword("myPassword");

try {
   xaConn2 = xads.getXAConnection(); // connect to dsn2
   xaConn3 = xads.getXAConnection(); // connect to dsn2
}
catch (SQLException e){
   e.printStackTrace();
   return;
}
```

Note: Once an **XAConnection** is established, autocommit is off.

Creating XAResource and Connection objects

After using **getXAConnection()** to obtain an **XAConnection**, you can use **getXAResource()** to obtain an **XAResource** object and **getConnection()** to obtain a **Connection** object.

Example 4.3

```
//get an XAResource
XAResource xaRes = null;
try {
   xaRes = xaConn.getXAResource();
}catch (SQLException e){
   e.printStackTrace();
   return;
}

//get an underlying physical Connection
Connection conn = null;
try {
   conn = xaConn.getConnection();
}catch (SQLException e){
   e.printStackTrace();
   return;
}
```

From this point, you can use the same connection, *conn*, for both local and global transactions. However:

- You must commit or rollback an active local transaction before starting a global transaction, otherwise you will get a XAER_OUTSIDE **XAException**.
- You must end an active global transaction before initiating a local transaction, otherwise you will get a SQLException “illegal combination of local transaction and global (XA) transaction”.

Registering a TimesTen XADataSource with the BEA WebLogic server

Note: Though TimesTen JTA has been demonstrated to work with the BEA WebLogic server on the Sun Solaris and Windows platforms, TimesTen cannot guarantee the operation of DTP software beyond the TimesTen implementation of JTA.

The general process of registering a data source with the BEA WebLogic server is described in *Managing JDBC Connectivity* in the *WebLogic Server Administration Guide*. URL:

```
http://edocs.bea.com/wls/docs61/adminguide/jdbc.html#1074927
```

To register a TimesTen JDBC driver with the WebLogic server, you need to:

- Set the CLASSPATH in the startWebLogic script file
- Configure a connection pool
- Configure a TxDataSource

The following sections describe each of these steps.

Set the CLASSPATH in the startWebLogic script file

Edit your WebLogic `startWebLogic.sh` (UNIX) or `startWeblogic.cmd` (Windows) script file to include the location of the TimesTen `\jdbc\lib\classes14.jar` file in your CLASSPATH.

For example, on NT:

```
set CLASSPATH=...;c:\TimesTen\TimesTen6.0\jdbc\lib\classes14.jar
```

Configure a connection pool

Configure a connection pool and include the URL of the TimesTen data store and the **TimesTenXaDataSource** driver class.

Example 4.4

```
Property Name          Property value
Name                   TimesTenXAPool
URL                    jdbc:timesten:dsn1
Driver Class Name      com.timesten.jdbc.xa.TimesTenXADataSource
initial capacity       1
maxCapacity            10
CapacityIncrement      1
Properties          url=jdbc:timesten:dsn1;user=user1;password=pwd1
```

Note: To accommodate all versions of WebLogic, you should specify the JDBC URL for both the URL and Properties.

Configure a TxDataSource

Configure a TxDataSource for your connection pool.

Example 4.5

```
Property Name            Property value
Name                     ttxads
targets                  myserver
jndiName                 jta.ttxads
poolname                 TimesTenXAPool
```

Error Handling

The XA specification has a limited, strictly defined set of errors that can be returned from XA interface calls. The ODBC **SQLError()** mechanism returns XA defined errors, along with any additional information.

The TimesTen XA related errors begin at number 11000. Errors 11002 through 11020 correspond to the errors defined by the XA standard.

See Chapter 6, “Warnings and Errors” for the complete list of errors.

5

Application Tuning

This chapter describes how to tune your Java application to run optimally on a TimesTen data store. For information on data store and SQL tuning, see Chapter 8, "Data Store Performance Tuning" in the *Oracle TimesTen In-Memory Database Operations Guide*.

This chapter includes general principles to consider when tuning Java applications for the Oracle TimesTen In-Memory Database and specific performance tuning tips for applications that utilize the JMS/XLA API.

Tuning Java applications

This section describes general principles to consider when tuning Java applications for TimesTen.

The topics described in this section are:

- Turn off autocommit mode
- Choose a timeout interval
- Choose the best method of locking
- Reduce contention
- Choose the appropriate logging options
- Prepare statements in advance
- Avoid unnecessary prepare operations
- Use the batch update facility for executing multiple statements
- Bulk fetch rows of TimesTen data
- Size transactions appropriately
- Use durable commits appropriately
- Use the ResultSet.getString method sparingly
- Avoid data type conversions
- Avoid transaction rollback
- Avoid frequent checkpoints

Turn off autocommit mode

By default, auto-commit is enabled, which forces a commit after each statement. Committing each statement after execution can have a significant negative impact on performance. For performance-sensitive applications, you may want to set auto-commit to off, as described in "Setting auto commit" on page 32.

The XACT_COMMITS column of the SYS.MONITOR table indicates the number of transaction commits.

If you do not include explicit commits in your application, the application can use up important resources unnecessarily, including memory and locks. All applications should do periodic commits.

Choose a timeout interval

To change the time-out interval for locks, use the built-in procedure **ttLockWait**. By default connections wait 10 seconds to acquire a lock. For some applications this interval may be either too long or too short.

Reduce contention

Data store contention can substantially impede application performance.

To reduce contention in your application:

- Choose the appropriate locking method. See "Choose the best method of locking" on page 79.
- Distribute data strategically in multiple tables and/or data stores.

If your application suffers a decrease in performance because of lock contention and a lack of concurrency, reducing contention is an important first step in improving performance.

The LOCK_GRANTS_IMMED, LOCK_GRANTS_WAIT and LOCK_DENIALS_COND columns in the SYS.MONITOR table provide some information on lock contention:

- LOCK_GRANTS_IMMED counts how often a lock was available and was immediately granted at lock request time.
- LOCK_GRANTS_WAIT counts how often a lock request was granted after the requestor had to wait for the lock to become available.
- LOCK_DENIALS_COND counts how often a lock request was not granted because the requestor did not want to wait for the lock to become available.

If limited concurrency results in a lack of throughput, or if response time is an issue, an application can serialize JDBC calls to avoid contention. This can be achieved by having a single thread issue all those calls. Using a single thread requires some queuing and scheduling on the part of the application, which has to trade off some CPU time for a decrease in contention and wait time. The result is

higher performance for low-concurrency applications that spend the bulk of their time in the data store.

Choose the best method of locking

When multiple connections access a data store simultaneously,TimesTen uses locks to ensure that the various transactions operate in apparent isolation. TimesTen supports the isolation levels described in Chapter 7, "Transaction Management and Recovery" in the *Oracle TimesTen In-Memory Database Operations Guide*. It also supports the locking levels: data store-level locking, table-level locking and row-level locking. You can use the **LockLevel** connection attribute to indicate whether data store-level locking or row-level locking should be used. Use the **ttOptSetFlag** procedure to set optimizer hints that indicate whether table locks should be used. The default lock granularity is row locks.

Choose an appropriate lock level

If there is very little contention on the data store, use table-level locking. It provides better performance and deadlocks are less likely. There is generally little contention on the data store when transactions are short and/or there are few connections. In that case, transactions are not likely to overlap.

Table-level locking is also useful when a statement accesses nearly all the rows on a table. Such statements can be queries, updates, deletes or multiple inserts done in a single transaction.

TimesTen uses table locks only with serializable isolation. If your application specifies table locks with any other isolation levels, TimesTen overrides table-level locking and uses row locks. However, the optimizer plan may still display table-level locking hints.

Data store-level locking restricts concurrency more than table-level locking, and is generally useful only for initialization operations, such as bulk-loading, when no concurrency is necessary. It has better response-time than row-level or table-level locking, at the cost of diminished throughput.

Row-level locking is generally preferable when there are many concurrent transactions that are not likely to need access to the same row.

Choose an appropriate isolation level

When using row-level locking, applications can run transactions at the SERIALIZABLE or READ_COMMITTED isolation level. The default isolation level is READ_COMMITTED. You can use the Isolation connection attribute to specify one of these isolation levels for new connections.

When running at SERIALIZABLE transaction isolation level, TimesTen holds all locks for the duration of the transaction, so:

- Any transaction updating a row blocks writers until the transaction commits.
- Any transaction reading a row blocks out writers until the transaction commits.

When running at READ_COMMITTED transaction isolation level, TimesTen only holds update locks for the duration of the transaction, so:

- Any transaction updating a row blocks out readers and writers of that row until the transaction commits.
- Phantoms are possible. A phantom is a row that appears during one read but not during another read, or appears in modified form in two different reads, in the same transaction, due to early release of read locks during the transaction.

You can determine if there is an undue amount of contention on your system by checking for time-out and deadlock errors (errors # 6001, 6002 and 6003). Information is also available in the LOCK_TIMEOUTS and DEADLOCKS columns of the SYS.MONITOR table.

Choose the appropriate logging options

The TimesTen Data Manager makes data store transactions durable by maintaining logs of transactions on disk. Log records are written when a transaction commits. Each commit incurs a disk write unless you specify non-durable commits or diskless logging, as discussed in "Use durable commits appropriately" on page 84.

With the default logging setting of **DurableCommits=1**, the log I/O is amortized over other concurrent connections using a technique called "group commit," response times may be long. Log I/O also affects throughput if there is not sufficient concurrency to hide the disk write. The LOG_FORCES column in the SYS.MONITOR table keeps track of the number of times the log was flushed to disk. Flushing the log to disk too frequently can result in I/O contention between the writes to the log and the checkpoint files. You can use the **LogDir** connection attribute to specify a different I/O path for log files.

If you set diskless logging, no log files are generated entire transactions may be rolled back, if the log buffer is not big enough. Hence transaction durability is not guaranteed by TimesTen, when diskless logging is used. Your application can guarantee durability by using checkpoints. In general, use diskless logging with caution as it may cause an excessive number of transaction aborts.

For some applications, lost transactions can be tolerated. For example, it may be possible to regenerate the data from another source in the event of a system or application failure, or the application may take checkpoints at strategic intervals to save data where needed. In these cases, it may be advantageous to turn off logging when connecting to the data store, as described in Chapter 1, "Data Store Attributes" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*. However, this can result in a lack of atomicity, as described in

Chapter 7, "Transaction Management and Recovery" in the *Oracle TimesTen In-Memory Database Operations Guide*.

Logs are also used to roll back transactions during error handling, as well as to redo transactions in case of application or system failure. If logging is turned off, most statements execute atomically, but entire transactions cannot be rolled back. For statements that do not execute atomically, TimesTen returns an error. If an application is in development or if it is susceptible to frequent rollbacks for other reasons, turning off logging may generate these errors, which may cause the data store to become inconsistent. In this case, it may be preferable to use diskless logging or logging to disk. To use non-durable commits, see "Use durable commits appropriately" on page 84.

Additional facts about logging are:

- Logging can be set to different values on a connection basis, but all concurrent connections must agree on the **Logging** attribute setting.
- Logging is required to use row-level locking.
- If logging is turned off or if diskless logging is enabled, you should include periodic non-durable commits in your application (see "Size transactions appropriately" on page 83). Durable commits are not permitted. (Because there is no log to write to disk, durability must be achieved through checkpoints.) If Logging is disabled, durable commits are not possible. If Logging is turned off, operations that are not atomic return an error or warning when the TimesTen Data Manager cannot restore the data store to its state prior to a failed operation.
- Logging to disk or diskless logging is required to cache Oracle tables.

Prepare statements in advance

As described in "Preparing SQL statements" on page 33, your application should prepare a statement in advance if it will be executed more than a few times. Make sure that the prepared statements have been committed in order to release locks held by the prepare and to allow the query plan to persist.

If you have applications that generate a statement multiple times searching for different values each time, use a parameterized statement to reduce compile time. For example, if your application generates statements like:

```
SELECT A FROM B WHERE C = 10
SELECT A FROM B WHERE C = 15
```

You can replace these statements with the single statement:

SELECT A FROM B WHERE C = ?

TimesTen shares prepared commands automatically, once it has been committed. As a result, an application's request to prepare a command for execution may be completed very quickly if a prepared version of the command already exists in

the system. Also, repeated requests for ***Statement.execute*** of the same command may be able to avoid the prepare overhead by sharing a previously prepared version of the command.

Even though TimesTen allows prepared statements to be shared, it is still a good practice, for performance reasons, to use parameterized statements. Using parameterized statements can reduce prepare overhead above and beyond what is made available through statement sharing.

Avoid unnecessary prepare operations

Because preparing SQL statements is an expensive operation, your application should minimize the number of calls to the **Connection.prepareStatement** method. Most applications prepare a set of commands at the beginning of a connection and use that set for the duration of the connection. This is a good strategy when connections are long, consisting of hundreds or thousands of transactions. But if connections are relatively short, a better strategy is to establish a long-duration connection that prepares the commands and executes them on behalf of all threads or processes. The trade-off here is between communication overhead and prepare overhead, and can be examined for each application. Prepared statements are invalidated when a connection is closed.

Use the batch update facility for executing multiple statements

The TimesTen JDBC driver supports the **addBatch**, **clearBatch**, and **executeBatch** methods for the **Statement** and **PreparedStatement** JDBC objects. These APIs can be used by applications to improve performance when multiple SQL update operations are executed through a **Statement** object or when multiple sets of parameters associated with a prepared statement are executed through a **PreparedStatement** object.

For **Statement** objects, a batch consists of a set of SQL write operation statements. Statements that return result sets are not allowed in a batch. A SQL write operation statement is added to a batch by calling the **addBatch** method. The set of SQL statements associated with a batch are executed through the **executeBatch** method. For example:

```
// turn off autocommit
conn.setAutoCommit(false);

Statement stmt = conn.createStatement();
stmt.addBatch("INSERT INTO employees VALUES (1000, 'Joe
Jones')");
stmt.addBatch("INSERT INTO departments VALUES (260, 'Shoe')");
stmt.addBatch("INSERT INTO emp_dept VALUES (1000, 260)");

// submit a batch of update commands for execution
int[] updateCounts = stmt.executeBatch();
conn.commit ();
```

For **PreparedStatement** objects, a batch consists of a set of prepared statement input parameters. Prepared statement parameters are added to the batch by executing **setXXX** calls followed by the **addBatch** call. The batch is executed via the **executeBatch** method. For example:

```
// turn off autocommit
conn.setAutoCommit(false);

PreparedStatement stmt = conn.prepareStatement(
"INSERT INTO employees VALUES (?, ?)");

// first set of parameters
stmt.setInt(1, 2000);
stmt.setString(2, "Kelly Kaufmann");
stmt.addBatch();

// second set of parameters
stmt.setInt(1, 3000);
stmt.setString(2, "Bill Barnes");
stmt.addBatch();

// submit the batch for execution
int[] updateCounts = stmt.executeBatch();
conn.commit ();
```

For more information on using the JDBC batch update facility see the Java Platform API specification for the **Statement** and objects.

Bulk fetch rows of TimesTen data

TimesTen provides an extension that allows an application to fetch multiple rows of data. For applications that retrieve large amounts of TimesTen data, fetching multiple rows can increase performance greatly. However, when using Read committed isolation, locks are held on all rows being retrieved until the application has received all the data, decreasing concurrency. For more information on this feature, see "Fetching multiple rows of data" on page 39.

Size transactions appropriately

Each transaction that generates log records (for example a transaction that does an INSERT, DELETE or UPDATE) by default incurs a disk write when the transaction commits. (See "Choose the appropriate logging options" on page 80.) Disk I/O affects response time and may affect throughput, depending on how effective group commit is.

Performance-sensitive applications should avoid unnecessary disk writes at commit. Use a performance analysis tool to measure the amount of time your application spends in disk writes (versus CPU time). If there seems to be an excessive amount of I/O, there are two steps you can take to avoid writes at commit:

- Adjust the transaction size.

- Adjust whether disk writes are performed at transaction commit. See "Use durable commits appropriately" on page 84.

Long transactions perform fewer disk writes per unit time than short transactions. However, long transactions also can reduce concurrency, as discussed in Chapter 7, "Transaction Management and Recovery" in the *Oracle TimesTen In-Memory Database Operations Guide*.

- If only one connection is active on a data store (for example, if it is an exclusive connection), longer transactions could improve performance. However, long transactions may have some disadvantages, such as longer rollbacks.
- If there are multiple connections, there is a trade-off between log I/O delays and locking delays. In this case transactions are best kept to their natural length, as determined by their requirements for atomicity and durability.

Use durable commits appropriately

By default, each TimesTen transaction results in a disk write at commit time. This practice ensures that no committed transactions are lost because of system or application failures. Applications can avoid some or all of these disk writes by performing non-durable commits. Non-durable commits do everything that a durable commit does except write the transaction log to disk. Locks are released and cursors are closed, but no disk write is performed.

Note: Some controllers or drivers only write data into cache memory in the controller or write to disk some time after the operating system is told that the write is completed. In these cases, a power failure may cause some information that you thought was durably committed to be lost. To avoid this loss of data, configure your disk to write to the recording media before reporting completion or use an Uninterruptable Power Supply (UPS).

The advantage of non-durable commits is a potential reduction in response time and increase in throughput. The disadvantage is that some transactions may be lost in the event of system failure. An application can force the log to disk by performing an occasional durable commit or checkpoint, thereby decreasing the amount of potentially lost data. In addition, TimesTen itself periodically flushes the log to disk when internal buffers fill up, limiting the amount of data that will be lost.

Transactions can be made durable or can be made to have delayed durability on a connection-by-connection basis. Applications can force a durable commit of a specific transaction by calling the **ttDurableCommit** procedure.

Applications that do not use non-durable commits can benefit from using synchronous writes in place of write and flush. To turn on synchronous writes set **LogFlushMethod**=**2**.

The XACT_D_COMMITS column of the SYS.MONITOR table indicates the number of transactions that were durably committed.

Use the ResultSet.getString method sparingly

Because Java strings are immutable, **ResultSet.getString** must allocate space for a new string (in addition to translating the underlying C-string to a Unicode string) each time it is called. Therefore, this is one of the costliest calls in JDBC.

In addition, you should not call **ResultSet.getString** on primitive numeric types, like Byte or Integer, unless it is absolutely necessary. It is much faster to call **ResultSet.getInt** on an integer column, for example.

Avoid data type conversions

TimesTen instruction paths are so short that even small delays due to data conversion can cause a relatively large percentage increase in transaction time.

Use the default **ResultSet.get***XXX* method for the data type of the data in the underlying database. For example, if the data type of the data is DOUBLE, to avoid data conversion in the JDBC driver you should call **getDouble**. For information on the default **get***XXX* method for a particular data type, refer to the "JDBC Quick Reference" in the book *JDBC Database Access with Java* by Hamilton, Cattell, Fisher. Similarly, use the default **PreparedStatement.set***XXX* method for the input parameter in an SQL statement. For example, if you are inserting data into a CHAR column using a **PreparedStatement**, you should use **setString**.

Avoid transaction rollback

When transactions fail due to erroneous data or application failure, they are rolled back by TimesTen automatically. In addition, applications often explicitly rollback transactions using **Connection.rollback()** to recover from deadlock or time-out conditions. This is not desirable from a performance point of view: a rollback consumes resources and the entire transaction is in effect wasted.

Applications should avoid unnecessary rollbacks. This may mean designing the application to avoid contention (see "Choose the best method of locking" on page 79) and checking application or input data for potential errors before submitting it, if possible. The XACT_ROLLBACKS column of the SYS.MONITOR table indicates the number of transactions that were rolled back.

Avoid frequent checkpoints

Applications that are connected to a data store for a long period of time occasionally need to call the **ttCkpt** procedure to checkpoint the data store so that log files do not fill up the disk. Transaction-consistent checkpoints can have

a significant performance impact because they require exclusive access to the data store.

It is generally best to call **ttCkpt** to perform a non-blocking (or "fuzzy") checkpoint than **ttCkptBlocking** to perform a blocking checkpoint. Non-blocking checkpoints may take longer, but they permit other transactions to operate against the data store at the same time and thus impose less overall overhead. You can increase the interval between successive checkpoints by increasing the amount of disk space available for accumulating log files.

As the log increases in size (if the interval between checkpoints is large), recovery time increases accordingly. If reducing recovery time after a system crash or application failure is important, frequent checkpoints may be preferable. The DS_CHECKPOINTS column of the SYS.MONITOR table indicates how often checkpoints have successfully completed.

Tuning JMS/XLA applications

This section contains specific performance tuning tips for applications that utilize the JMS/XLA API. JMS/XLA has some overhead that makes it not as fast as using the C XLA API. In the C API, records are returned to the user in a batch. In the JMS model an object is instantiated and each record is presented one at a time in a callback to the **MessageListener** **onMessage** method. High performance applications can use some tuning to overcome some of this overhead.

Configure xlaPrefetch parameter

The code underlying the JMS layer that reads the transaction log is more efficient if it can fetch as many rows as possible before presenting the object/rows to the user. The amount of prefetching is controlled in the jmsxla.xml configuration file with the "xlaPrefetch" parameter. Set the prefetch count to a large value like 100 or 1000.

Batch calls to ttXlaAcknowledge

Calls to **ttXlaAcknowledge** move the bookmark and involve updates to system tables, so one way to increase throughput is to wait until several transactions have been seen before issuing the call. This means that the reader application must have some tolerance for seeing the same set of records more than once. Moving the bookmark can be done manually using the **Session** **CLIENT_ACKNOWLEDGE** mode when instantiating a session:

```
Session session = connection.createSession
                  (false, Session.CLIENT_ACKNOWLEDGE);
```

For many applications, setting this value to 100 is a reasonable choice.

Increase log buffer size

A larger log buffer size is called for when using XLA. When XLA is turned on, additional log records are generated to store additional information for XLA. To ensure the log buffer is properly sized, one can watch for changes in the **SYS.MONITOR** table entries LOG_FS_READS and LOG_BUFFER_WAITS. For optimal performance, both of these values should remain 0. Increasing the log buffer size may be necessary to ensure the values remain 0.

6

Configuring TimesTen with Object/ Relational Persistence Frameworks

This chapter describes how to configure TimesTen with the Oracle TopLink and the Hibernate 3.0 object/relational persistence frameworks.

Configuring TimesTen with Oracle TopLink

This section summarizes the configuration steps required to use TimesTen with Oracle's TopLink object/relational persistence framework.

Configuring the TimesTen JDBC driver

After installing both TimesTen and TopLink, TopLink applications need access to the TimesTen JDBC driver `.jar` file and the native TimesTen shared libraries. You must:

1. Set the `CLASSPATH` environment variable to the location of the TimesTen JDBC driver `.jar` file. For example:

   ```
   % export CLASSPATH=/opt/TimesTen/tt60/lib/classes14.jar:$CLASSPATH
   ```

2. On UNIX platforms set the shared library path to the location of the TimesTen installation's `lib` directory. For example:


   ```
   % export LD_LIBRARY_PATH=/opt/TimesTen/tt60/lib:$LD_LIBRARY_PATH
   ```

   The TimesTen JDBC driver requires access to native libraries located in this directory.

TopLink Database Platform Support for TimesTen

TopLink version 10.1.3 includes direct runtime support for TimesTen SQL through a TimesTen specific platform class included in the TopLink distribution. In addition, the TopLink Mapping Workbench supports TimesTen-specific database type mappings. For more information on configuring TimesTen platform support in TopLink 10.1.3 refer to the *TopLink Application Developer's Guide*.

Configuring TopLink login properties

The **TopLink DatabaseLogin** class encapsulates the JDBC attributes associated with a TopLink session. This login class is automatically created when using a TopLink Mapping Workbench `project.xml` file.

Below is an example of a TimesTen login configuration from a `project.xml` file created by the Mapping Workbench. This login configuration enables parameter binding and JDBC batch updates for improved TimesTen performance.

```
<database-login>
<platform>oracle.toplink.internal.databaseaccess.TimesTenPlatform
</platform>
  <driver-class>com.timesten.jdbc.TimesTenDriver</driver-class>
  <connection-url>jdbc:timesten:TPTBM</connection-url>
  <user-name>scott</user-name>
  <password>tiger</password>
  <uses-native-sequencing>false</uses-native-sequencing>
  <sequence-preallocation-size>50</sequence-preallocation-size>
  <sequence-table>SEQUENCE</sequence-table>
  <sequence-name-field>SEQ_NAME</sequence-name-field>
  <sequence-counter-field>SEQ_COUNT</sequence-counter-field>
  <should-bind-all-parameters>true</should-bind-all-parameters>
  <should-cache-all-statements>true</should-cache-all-statements>
  <uses-byte-array-binding>true</uses-byte-array-binding>
  <uses-string-binding>false</uses-string-binding>
  <uses-streams-for-binding>false</uses-streams-for-binding>
  <should-force-field-names-to-upper-case>false
     </should-force-field-names-to-upper-case>
  <should-optimize-data-conversion>true
    </should-optimize-data-conversion>
  <should-trim-strings>true</should-trim-strings>
  <uses-batch-writing>false</uses-batch-writing>
  <uses-jdbc-batch-writing>true</uses-jdbc-batch-writing>
  <uses-external-connection-pooling>false
    </uses-external-connection-pooling>
  <uses-external-transaction-controller>false
    </uses-external-transaction-controller>
  <type>oracle.toplink.sessions.DatabaseLogin</type>
</database-login>
```

Configuring Hibernate with TimesTen 6.0

This section summarizes the configuration steps required to use TimesTen with the Hibernate object/relational persistence framework.

Configuring the TimesTen JDBC driver

After installing both TimesTen and Hibernate, Hibernate applications need access to the TimesTen JDBC driver `.jar` file and the native TimesTen shared libraries. You must:

1. Set the `CLASSPATH` environment variable to the location of the TimesTen JDBC driver jar file. For example:

   ```
   % export CLASSPATH=/opt/TimesTen/tt60/lib/classes15.jar:$CLASSPATH
   ```

   As an alternative to setting the `CLASSPATH` environment variable directly, the TimesTen JDBC driver `.jar` file can be copied into the `lib` directory of the Hibernate installation. When Hibernate initializes, all Java libraries in this directory will be automatically included in the `CLASSPATH`.

2. On UNIX platforms set the shared library path to the location of the TimesTen installation's `lib` directory. For example:


   ```
   % export LD_LIBRARY_PATH=/opt/TimesTen/tt60/lib:$LD_LIBRARY_PATH
   ```

   The TimesTen JDBC driver requires access to native libraries located in this directory.

Configuring Hibernate JDBC properties

Hibernate sessions can be configured using various methods. A common method is to set configuration properties in the `hibernate.properties` file. Hibernate searches for this file in a `root` directory of the `CLASSPATH`. Hibernate includes an example `hibernate.properties` file in the installation's `etc` directory.

To use TimesTen with Hibernate, these Hibernate properties should always be set as follows:

```
hibernate.dialect=org.hibernate.dialect.TimesTenDialect
```

```
hibernate.connection.driver_class=com.timesten.jdbc.TimesTenDriver
```

Connection specific properties are also required. For example to connect to a TimesTen DSN called MYDSN with the username of `scott` and password `tiger`, these properties would be set:

```
hibernate.connection.url=jdbc:timesten:MYDSN
hibernate.connection.username=scott
hibernate.connection.password=tiger
```

Hibernate includes additional JDBC driver properties that affect which functions in the driver are called by Hibernate. The following settings are recommended when using TimesTen:

```
# Connection.TRANSACTION_READ_COMMITTED isolation level
hibernate.connection.isolation=2

# a non-zero value enables the use JDBC batch updates
hibernate.jdbc.batch_size=32
hibernate.jdbc.use_streams_for_binary=false
hibernate.jdbc.use_get_generated_keys=false
hibernate.jdbc.use_scrollable_resultset=false
```

For debugging purposes, it may be useful to view the SQL generated by Hibernate and executed in TimesTen. To do this set the property:

```
hibernate.show_sql=true
```

Part II: TimesTen Java Reference

7

JDBC Reference

This appendix lists the JDBC interfaces supported by TimesTen and the TimesTen extensions to JDBC. The main topics are:

- Supported JDBC Interfaces
- TimesTen Extensions to JDBC

Supported JDBC Interfaces

This section lists all interfaces supported by the TimesTen implementation of JDBC. The packages described in this section are:

- java.sql support
- javax.sql.support

For complete reference information, see the Java documentation at

http://java.sun.com/j2se/1.4.2/docs/api/index.html

http://java.sun.com/j2se/1.5.0/docs/api/index.html

java.sql support

The supported java.sql interfaces are:

- **CallableStatement**
- **Connection**
- **DatabaseMetaData**
- **Driver**
- **ParameterMetaData**
- **PreparedStatement**
- **ResultSet**
- **ResultSetMetaData**
- **Statement**

The supported java.sql classes are:

- **Date**
- **DriverManager**
- **DriverPropertyInfo**
- **Time**
- **Timestamp**
- **Types**
- **DataTruncation**
- **SQLException**
- **SQLWarning**

In general, TimesTen does not support:

- Clob, blob, Array, Struct and Ref data types.
- Scrollable and updatable result sets.
- Result set holdability.
- Calendar for set/get Date.
- Calendar for set/get Time.

Restrictions for the specific interfaces are described below.

CallableStatement

TimesTen implements the **java.sql.CallableStatement** interface

Restrictions:

- You cannot pass parameters to CallableStatement objects by name. You must set parameters by ordinal numbers.
- You cannot use the SQL escape syntax.
- PreparedStatement pooling is not supported.

Connection

TimesTen implements the **java.sql.Connection** interface.

Restriction:

- No support for savepoints.

DatabaseMetaData

TimesTen implements the **java.sql.DatabaseMetaData** interface.

No restrictions.

Driver

TimesTen implements the **java.sql.Driver** interface from the classes:

`com.timesten.jdbc.TimesTenDriver`

`com.timesten.jdbc.TimesTenClientDriver`

No restrictions.

ParameterMetaData

TimesTen implements the **java.sql.ParameterMetaData** interface.

Restrictions:

- No support for **ResultSetMetaData getMetaData()**.
- The JDBC driver cannot determine whether or not a column is nullable and will always return *parameterNullableUnknown* from calls to **ParameterMetaData. isNullable(int param)**.
- The **ParameterMetaData.getScale()** returns 1 for VARCHAR, NVARCHAR and VARBINARY data types if they are INLINE. (See "Storage overhead" in the *Oracle TimesTen In-Memory Database API and SQL Reference Guide*.)

Note: Scale is of no significance to these data types.

PreparedStatement

TimesTen fully implements the **java.sql.PreparedStatement** interface.

No restrictions.

ResultSet

TimesTen implements the **java.sql.ResultSet** interface.

Restrictions:

- You cannot have multiple open ResultSet objects per statement.
- You cannot specify the holdability of a result set, so a cursor cannot remain open after it has been committed.

See "Working with result sets" on page 42 for more information.

ResultSetMetaData

TimesTen implements the **java.sql.ResultSetMetaData** interface

No restrictions.

Statement

TimesTen implements the **java.sql.Statement** interface

No restrictions.

javax.sql.support

- **ConnectionPoolDataSource**
- **PooledConnection**

ConnectionPoolDataSource

TimesTen implements the JDBC 3.0 **javax.sql.ConnectionPoolDataSource** factory for **javax.sql.PooledConnection** using:

```
com.timesten.jdbc.ObservableConnectionDS
```

PooledConnection

TimesTen implements the **javax.sql.PooledConnection** interface using:

```
com.timesten.jdbc.ObservableConnection
```

TimesTen Extensions to JDBC

The TimesTen extensions to JDBC are:

- TimesTenConnection
- TimesTenDataSource
- TimesTenXADataSource
- TimesTenVendorCode

TimesTenConnection

TimesTen implements the connection-level prefetch feature described in "Fetching multiple rows of data" on page 39 with the class:

`com.timesten.sql.TimesTenConnection`

The methods implemented by **TimesTenConnection** are:

- **getTtPrefetchClose**
- **getTtPrefetchCount**
- **isDataStoreValid**
- **setTtPrefetchClose**
- **setTtPrefetchCount**

getTtPrefetchClose

```
public boolean getTtPrefetchClose()
    throws SQLException;
```

Returns the current state of TT_PREFETCH_CLOSE.

Returns:	True if TT_PREFETCH_CLOSE is enabled, or false if disabled.
Throws:	SQLException - if a database access error occurs.

getTtPrefetchCount

```
public int getTtPrefetchCount()
    throws SQLException;
```

Returns the current prefetch count set for the TimesTen connection.

Returns:	The current prefetch count.
Throws:	SQLException - if a database access error occurs.

isDataStoreValid

```
public boolean isDataStoreValid()
    throws SQLException;
```

Detect whether or not the data store is valid.

Returns:	True if the data store is valid, or false if invalid.
Throws:	SQLException - if a database access error occurs.

setTtPrefetchClose

```
public void setTtPrefetchClose(boolean enable)
    throws SQLException;
```

Sets the state of TT_PREFETCH_CLOSE to true or false.

Parameters:	enable - set to true to enable TT_PREFETCH_CLOSE or false to disable.
Throws:	SQLException - if a database access error occurs.

setTtPrefetchCount

```
public void setTtPrefetchCount(int count)
    throws SQLException;
```

Establishes the number of rows to be prefetched for all of the statements on the TimesTen connection. Your application must have a direct driver connection to the data store to use this method. See "Fetching multiple rows of data" on page 39 for details.

Parameters:	count - The number of prefetches to set for the connection. The count can be any integer from 0 to 128, inclusive.
Throws:	SQLException - if a database access error occurs.

TimesTenDataSource

The **com.timesten.jdbc.TimesTenDataSource** class implements **javax.sql.DataSource.**

In addition to the **DataSource** methods, **TimesTenDataSource** also implements the methods:

- **getDescription**
- **getOraclePassword**
- **getPassword**
- **getUrl**
- **getUser**
- **setDescription**
- **setOraclePassword**
- **setPassword**
- **setUrl**
- **setUser**

getDescription

```
public String getDescription()
```

Returns the description of the data store set in the DSN. This description can be set either in by the **Description** DSN attribute or the **setDescription** method.

Returns:	The description of the data store, if present.

getOraclePassword

```
public String getOraclePassword()
```

Returns the password used by TimesTen Oracle Connect to connect to the Oracle database. This password can be set either in by the **OraclePWD** DSN attribute or the **setOraclePassword** method.

Returns: Password used to connect to an Oracle database, if present.

getPassword

```
public String getPassword()
```

Returns the password used to connect to TimesTen, if Access Control is enabled. This password can be set either in by the **PWD** DSN attribute or the **setPassword** method.

Returns: Password used to connect to TimesTen, if present.

getUrl

```
public String getUrl()
```

Returns the connection URL used by the **DriverManager.getConnection** method to establish a connection to a TimesTen data store. This URL is set by the **setUrl** method.

Returns: The current connection URL, if present.

getUser

```
public String getUser()
```

Returns the user ID used to connect to TimesTen, if Access Control is enabled. This user ID can be set either in by the **UID** DSN attribute or the **setUser** method.

Returns: User ID used to connect to TimesTen, if present.

setDescription

```
public void setDescription(String description)
```

Defines a description of the data store. The string specified by this method overrides the value of the **Description** DSN attribute and persists for the life of the **TimesTenDataSource** object.

Parameters: description - Description of the data store.

setOraclePassword

```
public void setOraclePassword(String oraPwd)
```

Sets the password used by TimesTen Oracle Connect to connect to the Oracle database. The password specified by this method overrides the value of the **OraclePWD** DSN attribute and persists for the life of the **TimesTenDataSource** object.

Parameters: `oraPwd` – Password used by Oracle Connect to establish a connection to Oracle.

setPassword

```
public void setPassword(String pwd)
```

Sets the password used to connect to TimesTen, if Access Control is enabled. The password specified by this method overrides the value of the **PWD** DSN attribute and persists for the life of the **TimesTenDataSource** object.

Parameters: pwd - Password used to establish a connection to TimesTen.

setUrl

```
public void setUrl (String url)
```

Sets the connection URL used by the **DriverManager.getConnection** method to establish a connection to a TimesTen data store.

You can also set DSN attributes in the URL For example, if Access Control is enabled, you can include the username and password needed to connect to the data store by calling:

```
setUrl("connectionURL;UID=user;PWD=passwd");
```

See "Create a connection URL for the data store" on page 29 for more information.

Parameters: `url` – URL used to establish a connection to TimesTen.

setUser

```
public void setUser(String user)
```

Sets the user ID used to connect to TimesTen, if Access Control is enabled. The user ID specified by this method overrides the value of the **UID** DSN attribute and persists for the life of the **TimesTenDataSource** object.

Parameters: `user` – User ID used to establish a connection to TimesTen.

TimesTenXADataSource

You can create a **javax.sql.XADataSource** object using the factory:

`com.timesten.jdbc.xa.TimesTenXADataSource`

See Chapter 4, "Distributed Transaction Processing: JTA" for usage information.

TimesTenVendorCode

The **TimesTenVendorCode** interface defines error names for TimesTen error numbers:

`com.timesten.jdbc.TimesTenVendorCode`

Table 7.1 TimesTen error names and numbers

Error Name	Error Number
TT_ERR_NONE	0
TT_ERR_HDRINSANE	400
TT_ERR_DBFILECLOSE	402
TT_ERR_DBREADLEN	403
TT_ERR_BLKHDRINSANE	404
TT_ERR_WRITELEN	406
TT_ERR_CKPTFILESEEK	407
TT_ERR_CKPTFILESYNC	408
TT_ERR_OSDESTROYDIR	411
TT_ERR_BADOPENMODE	412
TT_ERR_BADEXISTSMODE	413

TT_ERR_BADMISSINGMODE	414
TT_ERR_INCOMPATFILEMODES	415
TT_ERR_BLKMERGECONTENTION	416
TT_ERR_BADCONTROL2	417
TT_ERR_BADINTERNAL1	418
TT_ERR_BADCONTROL1	419
TT_ERR_CKPTNOTATTEMPTED	600
TT_ERR_CKPTFAILURE	601
TT_ERR_CKPTDENIED	602
TT_ERR_CKPTFUZZYDENIED	603
TT_ERR_CKPTLSNIGNORED	604
TT_ERR_CKPTLSNBAD	605
TT_ERR_CKPTBLOCKED	606
TT_ERR_BACKUPBLOCKED	607
TT_ERR_CKPTDBINVALID	609
TT_ERR_CKPTENDNOTFOUND	610
TT_ERR_GRPRSTMISMATCHED	620
TT_ERR_GRPRSTNONEXIST	621
TT_ERR_GRPRSTNOPERMCONN	622
TT_ERR_GRPRSTNOPERMCREA	623
TT_ERR_NOPERMACCESS	629
TT_ERR_LOGFILEPURGED	649
TT_ERR_BACKUPTYPE	650
TT_ERR_BACKUPPATHNOTINCR	651
TT_ERR_BACKUPDIRNOTFOUND	652
TT_ERR_INCBACKUPMAX	654
TT_ERR_INCBACKUPNOLOGGING	656

TT_ERR_BACKUPPATHLEN	657
TT_ERR_BACKUPHOLDBAD	658
TT_ERR_BACKUPSTALE	659
TT_ERR_BACKUPPTREQPURGED	660
TT_ERR_BACKUPPTREQFUTURE	661
TT_ERR_DBFILERESTOREUNFINISHED	662
TT_ERR_DISKLOGGINGREQUIRED	663
TT_ERR_FLUXWAITFAIL	700
TT_ERR_FLUXMARKFAIL	701
TT_ERR_DBBYPIDADDFAIL	702
TT_ERR_SUBDAEMONCONNFAIL	703
TT_ERR_PANICRECOVERY	704
TT_ERR_DBINUSEBYSAME	705
TT_ERR_THREADSPAWNFAIL	706
TT_ERR_MANUALLYUNLOADED	707
TT_ERR_MANUALLYLOADED	708
TT_ERR_DBICREATEFAIL	709
TT_ERR_DBCREATEINPROG	711
TT_ERR_MEMORYLOCK	712
TT_ERR_CONNECTASUSERSYS	713
TT_ERR_MISMATCHEDLOGDIR	714
TT_ERR_BADLOGDIR	715
TT_ERR_OSNOPERMDIR	720
TT_ERR_OSNOSUCHDIR	721
TT_ERR_LOGFLUSHER	722
TT_ERR_LOGCURSORNOTOPEN	723
TT_ERR_LOGCURSORNOCURRENT	724

TT_ERR_LOGCURSORUNPOSITIONED	725
TT_ERR_LOGCURSOREOF	726
TT_ERR_LOGDATALOSSPOSSIBLE	744
TT_ERR_PARTIALLOGBLK	745
TT_ERR_EXTRALOGFILES	746
TT_ERR_LOGRECINCOMPLETE	747
TT_ERR_SHORTLOGREDO	748
TT_ERR_LOGBLKINVALID	749
TT_ERR_LOGREADWRONGLSN	750
TT_ERR_LOGREADBADLEN	751
TT_ERR_LOGREADBADTYPE	752
TT_ERR_LOGCOPYFILEOPEN	754
TT_ERR_LOGCOPYFILECHMOD	755
TT_ERR_LOGCOPYFILEWRITE	756
TT_ERR_LOGFILEHDR	759
TT_ERR_LOGREADEOF	760
TT_ERR_LOGFILESIZE	761
TT_ERR_INVALIDTRUNCLSN	762
TT_ERR_LOGFILETRUNCATE	763
TT_ERR_LOGFILEMOVE	764
TT_ERR_LOGCOPYDIRCREATE	765
TT_ERR_LOGRESFILECREATE	766
TT_ERR_LOGRESFILECHMOD	767
TT_ERR_LOGRESFILESZ	768
TT_ERR_LOGRESFILESZBAD	769
TT_ERR_LOGRESFILEOPEN	770
TT_ERR_LOGRESFILEWRITE	771

TT_ERR_LOGRESFILESEEK	772
TT_ERR_LOGRESFILESYNC	773
TT_ERR_LOGRESFILECLOSE	774
TT_ERR_LOGRESFILEDELETE	775
TT_ERR_LOGRESFILERENAME	776
TT_ERR_LOGRESEXHAUSTED	777
TT_ERR_LOGFSFULL	778
TT_ERR_LOGXACTRLBK	779
TT_ERR_LOGFILEMISSING	780
TT_ERR_LOGFILESYNC	781
TT_ERR_LOGFILERENAME	782
TT_ERR_LOGFILESYNCRETRY	783
TT_ERR_LRTYPEDISALLOWED	784
TT_ERR_NOLRATLSN	785
TT_ERR_LOGRESFILETRUNCATE	786
TT_ERR_LOGWRITELEN	790
TT_ERR_SUBDAEMONDEAD	791
TT_ERR_LOGREADLEN	792
TT_ERR_NOSUBDAEMONS	793
TT_ERR_JOINFAIL	794
TT_ERR_PROTOCOL	795
TT_ERR_PROCINIT	796
TT_ERR_REENTRANT	797
TT_ERR_DAEMONFAILURE	798
TT_ERR_DAEMONCONNECT	799
TT_ERR_TOOMANYERRORS	800
TT_ERR_MALLOCFAILED	801

TT_ERR_DBALLOCFAILED	802
TT_ERR_OBJNAMETOOLONG	803
TT_ERR_UNKNOWN	804
TT_ERR_UNIMPLEMENTED	805
TT_ERR_GARBAGELEFTONHEAP	806
TT_ERR_NULLPOINTER	807
TT_ERR_ATOMIC	808
TT_ERR_HPPGDIROVERFLOW	809
TT_ERR_DBPOINTER	810
TT_ERR_FORCEDERROR	811
TT_ERR_BADPLATFORM	812
TT_ERR_DBMINSZ	814
TT_ERR_SBINITFAIL	817
TT_ERR_DBMAXSZ	818
TT_ERR_BADDBRELEASE	819
TT_ERR_LOGSDESTROY	820
TT_ERR_DBNOCKPT	821
TT_ERR_COMPACT	823
TT_ERR_ERRORLOST	824
TT_ERR_DBBADTEMP	826
TT_ERR_DBBADLOGFLAG	827
TT_ERR_DBBADCOPYFLAG	829
TT_ERR_DBFILECREATE	830
TT_ERR_DBFILECHMOD	831
TT_ERR_DBFILEOPENRD	832
TT_ERR_DBNOTFOUND	833
TT_ERR_DBFILEINVALID	834

TT_ERR_DBFILECKPTUNFINISHED	835
TT_ERR_DBSHMCREATE	836
TT_ERR_DBSHMATTACH	837
TT_ERR_DBSHMGET	838
TT_ERR_DBINUSE	839
TT_ERR_DBMALLOCCREATE	840
TT_ERR_DBSHMDETACH	841
TT_ERR_DBIDINVALID	842
TT_ERR_NOTSUPPORTED	843
TT_ERR_DBFILEWRITE	844
TT_ERR_DBFILEREAD	845
TT_ERR_BADCONNECT	846
TT_ERR_DBFILEOPENWT	847
TT_ERR_RECOVERYFAILED	848
TT_ERR_RECOVERYNOSPACE	849
TT_ERR_DBISCONNECTED	850
TT_ERR_DBFILEDESTROY	851
TT_ERR_MEMORYLOCKWARN	852
TT_ERR_DBNOLOGGING	853
TT_ERR_DBEXISTS	854
TT_ERR_TOPLVLHPNOTDEFINED	855
TT_ERR_DBFILESZGET	856
TT_ERR_BADHPCREATEFLAGS	857
TT_ERR_XACTSVPTBAD	860
TT_ERR_XACTMAX	862
TT_ERR_XACTREQUIRED	863
TT_ERR_XACTEXISTS	864

TT_ERR_LOGFILEDESTROY	865
TT_ERR_OSBUFFLUSH	867
TT_ERR_OSOPENDIR	868
TT_ERR_OSUNLINK	869
TT_ERR_DBSHMCLOSE	870
TT_ERR_NULLCOL	871
TT_ERR_TUPLENILMAX	872
TT_ERR_TUPLENILROWMAX	873
TT_ERR_MISSINGCOLINFO	875
TT_ERR_OVERLAPCOLINFO	876
TT_ERR_ALIGNCOLINFO	877
TT_ERR_BADCOLNUM	878
TT_ERR_IXPRIMARYEXISTS	879
TT_ERR_IXMAXPERTABLE	880
TT_ERR_HASHKEYNEQPRIMKEY	881
TT_ERR_IXPRIMARYUNIQUE	882
TT_ERR_INDEXHASCURSORS	883
TT_ERR_BADHASHINDEX	884
TT_ERR_BADTTREEINDEX	885
TT_ERR_IXOPUNSUPPORTED	886
TT_ERR_BADIXPRIMARY	887
TT_ERR_LOGFILEBAD	888
TT_ERR_LOGFILENUMBERMAX	889
TT_ERR_LOGFILENAMETOOLONG	890
TT_ERR_LOGFILEEXISTS	891
TT_ERR_LOGFILECREATE	892
TT_ERR_LOGFILEOPEN	893

TT_ERR_LOGFILEWRITE	894
TT_ERR_LOGFILECLOSE	895
TT_ERR_LOGFILEREAD	896
TT_ERR_LOGFILESEEK	897
TT_ERR_LOGRECTOOLONG	898
TT_ERR_LOGBUFSZBAD	899
TT_ERR_TEMPFILENAMEGEN	900
TT_ERR_RCLMFILECREATE	901
TT_ERR_RCLMFILEWRITE	902
TT_ERR_RCLMFILESEEK	903
TT_ERR_RCLMFILEREAD	904
TT_ERR_BADINDEXID	905
TT_ERR_LOGFILECHMOD	906
TT_ERR_KEYEXISTS	907
TT_ERR_KEYNOTFOUND	908
TT_ERR_KEYNOTUPDATABLE	909
TT_ERR_TUPISDELETED	910
TT_ERR_NULLTBLNAME	911
TT_ERR_TUPNOTINTBL	912
TT_ERR_BADMAXCNT	913
TT_ERR_BADTABLENAME	914
TT_ERR_BADPTNNUM	915
TT_ERR_KEYCOLNOTNULL	916
TT_ERR_BADKEYCOLNUM	917
TT_ERR_BADKEYCOLCNT	918
TT_ERR_BADCMPCOND	919
TT_ERR_BADDEFAULTVAL	920

TT_ERR_COLTYPEUNSUP	921
TT_ERR_BADTUPLEID	922
TT_ERR_BADTUPIMPL	923
TT_ERR_BADTUPIMPLFOROP	924
TT_ERR_DBSEMCREATE	925
TT_ERR_DBLATCHCREATE	926
TT_ERR_DBLATCHOPEN	927
TT_ERR_DBSEMERROR	928
TT_ERR_BADLOGCONNECTFLAGS	929
TT_ERR_DBMAXCONNECTS	931
TT_ERR_INVALIDCOLUMNNAME	932
TT_ERR_SQLBADSTRING	933
TT_ERR_SQLBADBINARY	934
TT_ERR_SQLSTMTRETURNSNORESULT	935
TT_ERR_SQLSTMTRETURNSMULTRESULTS	936
TT_ERR_INVALIDSQLCNUM	937
TT_ERR_SQLSTMTREQUIRESARGS	938
TT_ERR_SQLSTMTREQUIRESNOARGS	939
TT_ERR_SQLCMDMAXRES	940
TT_ERR_SCANSACTIVE	941
TT_ERR_IMPRECISENUMCOERCION	942
TT_ERR_ZEROMAXTUPS	943
TT_ERR_SQLCMDNOTUSABLE	946
TT_ERR_LOGFILENAMEGEN	947
TT_ERR_RDLOGBUFSZBAD	948
TT_ERR_BADFORMATWIDTH	950
TT_ERR_BADFORMATPRECISION	951

TT_ERR_BADFORMATNULLSTRING	952
TT_ERR_SQLCMDBADVERNUM	954
TT_ERR_STRINGLONG	960
TT_ERR_BADVALUE	962
TT_ERR_CKPTLOGWRITE	963
TT_ERR_LOGTRUNCATE	964
TT_ERR_JOINORDERINAPPLICABLE	965
TT_ERR_JOINSTRINGLONG	966
TT_ERR_INDOPTINVALID	968
TT_ERR_INDOPTSTRINGLONG	969
TT_ERR_BADCURHANDLE	970
TT_ERR_BUFTOOSMALL	971
TT_ERR_BADDBHANDLE	972
TT_ERR_BADSVPTHANDLE	973
TT_ERR_BADFMTHANDLE	974
TT_ERR_BADTBLHANDLE	975
TT_ERR_BADIXHANDLE	976
TT_ERR_BADTUPHANDLE	977
TT_ERR_BADCMDHANDLE	978
TT_ERR_BADCMDARGTUPHANDLE	979
TT_ERR_BADBUFALIGNMENT	980
TT_ERR_BADCURTYPE	981
TT_ERR_STRINGLONGERR	982
TT_ERR_INDOPTINVALIDNAME	983
TT_ERR_INDOPTINAPPLICABLE	984
TT_ERR_NOSTATSONSTATSTABLES	985
TT_ERR_DSKLSSLOGBUFOVFLW	986

TT_ERR_DSKLSSLOGRECTOOLONG	987
TT_ERR_DSKLSSLOGXACTRLBK	988
TT_ERR_DBBADDSKLSSLOGFLAG	989
TT_ERR_TABLEBADMAXCNT	990
TT_ERR_QUOTAOVERMAXCNT	991
TT_ERR_TABLEOVERQUOTA	992
TT_ERR_INDEXOVERQUOTA	993
TT_ERR_DRTYBYTE	994
TT_ERR_DBSHMDESTROY	995
TT_ERR_BACKUPXACTMISSING	996
TT_ERR_DSPATHTOOLONG	997
TT_ERR_BADSYNTAX	1001
TT_ERR_IDENTIFIERTOOLONG	1002
TT_ERR_NEWLINEIDENTIFIER	1003
TT_ERR_ZEROLENIDENTIFIER	1004
TT_ERR_SPACESIDENTIFIER	1005
TT_ERR_ORDERBYCOLMUSTBEPOS	1017
TT_ERR_CONSTNUMNOTREPRESENTABLE	1022
TT_ERR_TABLENOTINFROMLIST	1024
TT_ERR_NUMINSERTCOLSANDVALSNOTSAME	1025
TT_ERR_COLNAMEBLANK	1026
TT_ERR_FLOATPRECISIONINVALID	1027
TT_ERR_AGGFUNCCANTBENESTED	1032
TT_ERR_MULTPRIMARYKEYSPEC	1061
TT_ERR_STALETBLHANDLE	1102
TT_ERR_SYSDATECONVERTTOTS	1103
TT_ERR_SYSDATEGETTIME	1104

TT_ERR_INVALIDCOLREF	1105
TT_ERR_XACTIDNOTFOUND	1200
TT_ERR_XACTIDROLLBACK	1201
TT_ERR_XACTIDROLLBACKSET	1203
TT_ERR_XACTIDROLLBACKXA	1204
TT_ERR_INTERVALINVL	2042
TT_ERR_BINARYLONG	2043
TT_ERR_BINARYLONGERR	2044
TT_ERR_TABLEDOESNOTEXIST	2206
TT_ERR_TABLEALREADYEXISTS	2207
TT_ERR_COLUMNNOTINTABLE	2208
TT_ERR_DUPLICATECOLUMN	2209
TT_ERR_COLUMNAMBIGUOUS	2210
TT_ERR_COLUMNNOTFOUND	2211
TT_ERR_INDEXNOTINTABLE	2212
TT_ERR_DUPLICATEINDEX	2213
TT_ERR_ROWIDCOLUMN	2214
TT_ERR_DROPPRIMARYINDEX	2215
TT_ERR_LONGINDEXKEYLEN	2216
TT_ERR_DUPLICATECOLININSERT	2221
TT_ERR_INDEXNAMEAMBIGUOUS	2222
TT_ERR_INDEXNOTFOUND	2223
TT_ERR_DUPLICATECOLINUPDATE	2249
TT_ERR_ADDCOLDUPLICATECOLNAME	2250
TT_ERR_CONSTRAINTCOLNOTDEFINED	2286
TT_ERR_COLMUSTBENOTNULL	2294
TT_ERR_KEYCOLDUP	2295

TT_ERR_DUPLICATETBLINFROMCL	2376
TT_ERR_SELFINSERT	2377
TT_ERR_TOOMANYCOLS	2400
TT_ERR_PRIMARYKEYCOLS	2404
TT_ERR_TOOMANYCOLSININDEX	2405
TT_ERR_COLLENGTHTOOLONG	2407
TT_ERR_COLLENGTHTOOSHORT	2414
TT_ERR_TOOMANYHASHCOLS	2416
TT_ERR_TOOMANYVALUES	2420
TT_ERR_COLNUMEXCEEDSSELLISTLEN	2421
TT_ERR_BCDTYPEPRECISIONINVALID	2430
TT_ERR_BCDTYPESCALEINVALID	2431
TT_ERR_SCALELARGERTHANPRECISION	2432
TT_ERR_FIXEDPTNUMINVALID	2433
TT_ERR_DECIMALOVERFLOW	2434
TT_ERR_DECIMALLONG	2435
TT_ERR_DECIMALDIVIDEBYZERO	2436
TT_ERR_DECIMALMULTSCALEOVERFLOW	2437
TT_ERR_BCDVALPRECISIONINVALID	2438
TT_ERR_BCDVALSCALEINVALID	2439
TT_ERR_DECIMALOVERFLOWCOERCE	2440
TT_ERR_INTEGEROVERFLOW	2600
TT_ERR_INTEGERDIVIDEBYZERO	2602
TT_ERR_EXTPRECISIONOVERFLOW	2603
TT_ERR_EXTPRECISIONDIVIDEBYZERO	2604
TT_ERR_EXTPRECISIONUNDERFLOW	2605
TT_ERR_CHARDATECONVERSION	2606

TT_ERR_CHARTIMECONVERSION	2607
TT_ERR_CHARTIMESTAMPCONVERSION	2608
TT_ERR_INCOMPATIBLETYPES	2609
TT_ERR_BADOPNDDATATYPEFOROP	2610
TT_ERR_DATECHARCONVERSION	2611
TT_ERR_TIMECHARCONVERSION	2612
TT_ERR_TIMESTAMPCHARCONVERSION	2613
TT_ERR_INTEGEROVERFLOWCOERCE	2614
TT_ERR_COLNOTINGROUPBYLIST	2705
TT_ERR_ORDERBYEXPNOTINDISTINCTLIST	2706
TT_ERR_CANTUPDATEHASHKEY	2712
TT_ERR_MULTDISTINCTCOLSINAGGFUNS	2763
TT_ERR_DISTINCTAGGEXP	2764
TT_ERR_INVALIDAGGEXP	2765
TT_ERR_BADCHARAFTERESC	2774
TT_ERR_BADESCCLINLIKE	2775
TT_ERR_PARAMSASBOTHBINOPARGS	2776
TT_ERR_PARAMASUNOPARG	2777
TT_ERR_CANTINFERPARAMTYPE	2778
TT_ERR_DATALEN	2779
TT_ERR_BADPAGESVALUE	2781
TT_ERR_INVALIDALTERTABLE	2783
TT_ERR_INVLDEFAULTUSE	2784
TT_ERR_INVLDROPUNIQ	2785
TT_ERR_INVLADDUNIQ	2786
TT_ERR_DATETIMEARITH	2787
TT_ERR_INTERVALOVERFLOWCOERCE	2788

TT_ERR_DATETIMEARITHNOTYPE	2789
TT_ERR_CANTINFERCASEEXPTYPE	2790
TT_ERR_JOINUPDATESYNTAX	2791
TT_ERR_JOINUPDATEINVLCOL	2792
TT_ERR_COLROWIDREF	2793
TT_ERR_ARGOUTOFRANGE	2794
TT_ERR_INVLUNIONORDERBY	2795
TT_ERR_INVLONCOMMITOPT	2796
TT_ERR_PARAMSASFIRSTINSTRARGS	2797
TT_ERR_PARAMSASFIRSTSUBSTRARG	2798
TT_ERR_BADCHARACTER	2903
TT_ERR_BADELEMNUMFORMAT	2904
TT_ERR_BADELEMDATEFORMAT	2905
TT_ERR_BADNUMFORMAT	2906
TT_ERR_BADDATEFORMAT	2907
TT_ERR_DECIMALCHARCONVERSION	2908
TT_ERR_NUMERICCHARCONVERSION	2909
TT_ERR_TOCHARFLOATOVERFLOW	2910
TT_ERR_TOCHARFLOATUNDERFLOW	2911
TT_ERR_INVALIDUNICODEESCAPE	2912
TT_ERR_CONTRAINTNOTINTABLE	2998
TT_ERR_FOREIGNCIRCULARREF	2999
TT_ERR_FOREIGNSELFREF	3000
TT_ERR_FOREIGNKEYMATCH	3001
TT_ERR_FOREIGNKEYLOG	3002
TT_ERR_FOREIGNKEYINUSE	3003
TT_ERR_FOREIGNKEYCOLMISMATCH	3004

TT_ERR_FOREIGNKEYIDXMATCH	3005
TT_ERR_FOREIGNKEYSYSTBL	3006
TT_ERR_FOREIGNKEYTWONULLS	3007
SB_SCALARSUBQUERYRETURNDUPLICATES	3008
TT_ERR_UNIQCONSTINDEX	3009
TT_ERR_FKEYCOLDUP	3011
TT_ERR_FKEYCACHEONLY	3012
TT_ERR_CACHEONLY	3013
TT_ERR_ILLEGALORWITHOJ	3100
TT_ERR_OJNOTALLOWEDINSELCLAUSE	3101
TT_ERR_OJINNERTABW1OUTERTAB	3102
TT_ERR_OJWITHTWOTABNOTALLOWED	3103
TT_ERR_OJREFONLYONEOJ	3104
TT_ERR_POSITIVEFIRSTNVALUEREQ	3105
TT_ERR_FIRSTNINSUBQ	3106
TT_ERR_ROWSMTONSPEC	3107
TT_ERR_ILLEGALJOINEDTABWITHOJ	3108
TT_ERR_MVEXPRESSIONWITHNONAME	3110
TT_ERR_MVDUPLICATECOLUMNNAME	3111
TT_ERR_MVUSEDROPVIEW	3112
TT_ERR_MVDROPTABLE	3113
TT_ERR_MVALTERVIEWTABLE	3114
TT_ERR_MVALTERTABLE	3115
TT_ERR_MVDETAILTABLEREP	3116
TT_ERR_MVSQLTOOBIG	3117
TT_ERR_VIEWEXISTS	3118
TT_ERR_MVTOOMANYVIEWS	3119

TT_ERR_VIEWNOTEXISTS	3120
TT_ERR_MVDETAILTABLECACHEGROUP	3121
TT_ERR_MVILLEGLFKONVIEW	3123
TT_ERR_MVSYSTEMTABLE	3124
TT_ERR_MVDEFTRUNCATED	3125
TT_ERR_MVILLEGALNBDISO	3126
TT_ERR_MVILLEGALNOLOG	3127
TT_ERR_NMVILLEGALCOLNAMELIST	3128
TT_ERR_NMVILLEGALCOLNAME	3129
TT_ERR_NMVILLEGALVIEWSPEC	3130
TT_ERR_DERIVEDTABILLEGALNAME	3131
TT_ERR_USEDROPVIEW	3132
TT_ERR_MVILLEGALVIEWUSE	3133
TT_ERR_SEQINCRZERONOTALLOWED	3200
TT_ERR_SEQCACHENOTALLOC	3201
TT_ERR_SEQILLEGALMAXVALUE	3202
TT_ERR_SEQNOTFOUND	3204
TT_ERR_SEQDUPLICATE	3205
TT_ERR_SEQVALNOTALLOWEDHERE	3206
TT_ERR_SEQVALNOTALLOWEDINSUBQ	3207
TT_ERR_SEQVALNOTALLOWINWHERE	3208
TT_ERR_SEQVALNOTSUPPINVIEW	3209
TT_ERR_SEQLIMITREACHED	3210
TT_ERR_GETCURRVALFAILED	3211
TT_ERR_GETNEXTVALFAILED	3212
TT_ERR_LOADSEQTOCACHEFAIL	3213
TT_ERR_ADDSEQVALINFOFAIL	3214

TT_ERR_CLEANUPSEQCACHERLBK	3215
TT_ERR_STATSSPEC	3221
TT_ERR_INTERNAL	4053
TT_ERR_WARNINTERNAL	4054
TT_ERR_EVENTSEEDING	4055
TT_ERR_CKPTINNFS	4501
TT_ERR_LOGSINNFS	4502
TT_ERR_GENARGRNGBAD	4600
TT_ERR_GENARGVALBAD	4601
TT_ERR_CACHENOAGENT	5002
TT_ERR_CACHEBADTABLENAMEREF	5005
TT_ERR_CACHESQLSTRINGTOOLONG	5006
TT_ERR_CACHEAGENT	5007
TT_ERR_CACHEODBCERROR	5009
TT_ERR_CACHENOORAID	5010
TT_ERR_CACHEOCIERROR	5011
TT_ERR_CACHEOCILOGINFAILED	5012
TT_ERR_CACHENOTCONNECTED	5013
TT_ERR_CACHEDEFININGSQL	5014
TT_ERR_CACHECOLMISMATCH	5015
TT_ERR_CACHETOOMANYCONNS	5016
TT_ERR_CACHEBADTABLENAME	5019
TT_ERR_CACHEDATATRUNCATED	5020
TT_ERR_CACHEGETORAMETADATA	5021
TT_ERR_CACHEOUTFILEOPEN	5022
TT_ERR_CACHEORAFAILTTROLLBACK	5025
TT_ERR_CACHENOALTERTABLE	5026

TT_ERR_CACHEPOPWARNS	5029
TT_ERR_CACHEAGENTBUFFFULL	5030
TT_ERR_CACHESHARESTRANS	5034
TT_ERR_CACHENUMTABLE	5035
TT_ERR_CACHELOADWARNS	5036
TT_ERR_CACHELOADGROUP	5037
TT_ERR_CACHEREFRESHWARNS	5038
TT_ERR_CACHEREFRESHGROUP	5039
TT_ERR_CACHEBADLOCKLEVEL	5045
TT_ERR_CACHEMUSTCOMMIT	5046
TT_ERR_CACHENOTALLOWED	5047
TT_ERR_FEATURENOTSUPPORTED	5048
TT_ERR_CACHEDEFTRUNCATED	5050
TT_ERR_MSGTOOAFAILED	5051
TT_ERR_CACHEGROUPNAMEREQD	5052
TT_ERR_CACHEILLEGLFKOP	5053
SB_WARNCACHEORAROLLBACK	5054
TT_ERR_CACHEORAFORCEFAILED	5055
TT_ERR_GENLIBNAME	5101
TT_ERR_CACHELOADLIB	5102
TT_ERR_CACHEUNLOADLIB	5103
TT_ERR_CACHEINITFP	5104
TT_ERR_OCIINITFAILED	5105
TT_ERR_ALLOCHANDLEFAILED	5106
TT_ERR_CACHEOCIERROR2	5107
TT_ERR_BDBCURSORNOTOPEN	5108
TT_ERR_BDBMACROERR	5109

TT_ERR_CACHENOORATABLE1	5110
TT_ERR_CACHENOKEY1	5111
TT_ERR_CACHEORAVARCHAR	5112
TT_ERR_CACHEORAFLOAT	5113
TT_ERR_CACHETYPEMAPKEY	5114
TT_ERR_CACHETYPEMAP	5115
TT_ERR_CACHENOCOLUMN	5116
TT_ERR_CACHETBLWHERE	5117
TT_ERR_CACHEPREC	5118
TT_ERR_CACHENULL	5119
TT_ERR_CACHENOKEYWARN	5120
TT_ERR_CACHETYPEWARN	5121
TT_ERR_CACHENOKEYERR	5123
TT_ERR_CACHERESTRICT	5124
TT_ERR_CACHEORATIMESTAMP	5125
TT_ERR_CACHERESTRICTTYPED	5126
TT_ERR_BDBCONNECTINVALID	5127
TT_ERR_BDBHANDLEINVALIDRETRY	5128
TT_ERR_BDBHANDLEINVALIDREPREPARE	5129
TT_ERR_CACHESQLERROR	5130
TT_ERR_BDBCONNECTERROR	5131
TT_ERR_BDBHANDLEINVALIDREPREPAREWARN	5132
TT_ERR_BDBSETORASIGINTFAILED	5133
TT_ERR_UNABLETOCHKORASERVERVER	5134
TT_ERR_LOGMNRNOTSUPPORTED	5135
TT_ERR_TBLDEFINITIONNEW	5136
TT_ERR_CACHENOORASYNONYM1	5137

TT_ERR_CACHESYNONYMWITHDBLINK	5138
TT_ERR_CACHESYNONYMNOBASEOWNER	5139
TT_ERR_CACHENOORAOBJECT1	5140
TT_ERR_CACHEORAOBJNOTSUPPORTED	5141
TT_ERR_CACHEHASSYNONYM	5142
TT_ERR_CACHESYNONYMTOODEEP	5143
TT_ERR_CACHEHASMATVIEWWARN	5144
TT_ERR_CACHEMATVIEWRESTRICTED	5145
TT_ERR_CACHENOPUBLICSYNONYMS	5146
TT_ERR_CACHEHASSYNONYMWARN	5147
TT_ERR_CACHENOCORRORATABLE1	5148
TT_ERR_ORACLEPWD	5149
TT_ERR_BDBSTMTSTATEINVALID	5150
TT_ERR_BDBSTMTTYPENOTSUPPORTED	5151
TT_ERR_BDBSTMTTYPEUNKNOWN	5152
TT_ERR_BDBSTMTOPNOTSUPPORTED	5153
TT_ERR_BDBSTMTWRONGNUMPARAMS	5155
TT_ERR_BDBSTMTTOOMANYPARAMS	5156
TT_ERR_BDBORATIMESTAMP	5157
TT_ERR_BDBORACOMPILEERR	5158
TT_ERR_UNIQINDEXMADENONUNIQ	5159
TT_ERR_LOCKLOGTRUNC	5160
TT_ERR_BIPROPLOGMARKING	5161
TT_ERR_ORAAGENTTEMP	5201
TT_ERR_ORACLEHOME	5202
TT_ERR_ORACLELIB	5203
TT_ERR_PATHENVVAR	5204

TT_ERR_OCINOTINPATH	5205
TT_ERR_ORACLEHOMELEN	5206
TT_ERR_ORACONSTRAINTUNIQUE	5210
TT_ERR_ORAOUTOFRESOURCE	5211
TT_ERR_ORANOLONGERCONNECTED	5212
TT_ERR_ORABADLOGIN	5213
TT_ERR_ORACANNOTCONNECT	5214
TT_ERR_ORARESOURCETIMEOUT	5215
TT_ERR_ORARESOURCEBUSY	5216
TT_ERR_ORADEADLOCK	5217
TT_ERR_ORAPARAMETERBAD	5218
TT_ERR_ORACONNECTFAILEDTMP	5219
TT_ERR_ORACONNECTFAILEDPERM	5220
TT_ERR_ORASYNTAXERR	5221
TT_ERR_ORACONSTRAINT	5222
TT_ERR_ORACONSTRAINTORPHAN	5223
TT_ERR_ORACONSTRAINTHASCHILD	5224
TT_ERR_ORARECOVERYINPROG	5225
TT_ERR_ORATRUNC	5226
TT_ERR_MASTERDSNAME	5240
TT_ERR_USINGAWTREPSCHEMA	5241
TT_ERR_AWTREPSCHEMAGENFAILED	5242
TT_ERR_OCNOREPAGENTERR	5243
TT_ERR_OCNOREPAGENTWARN	5244
TT_ERR_OCREPAGENTUPERR	5245
TT_ERR_OCREPAGENTUPWARN	5246
TT_ERR_OCORAAGENTUPERR	5247

TT_ERR_AWTONDISKLESSENV	5248
TT_ERR_AWTINITFAILED	5249
TT_ERR_AWTINITFAILED0	5250
TT_ERR_AWTNEEDSADMINUIDPWD	5251
TT_ERR_AWTGETUIDFAILED	5252
TT_ERR_AWTBEHIND	5253
TT_ERR_AWTNOTFOUND	5254
TT_ERR_AWTFAILTHRESHOLDNEG	5255
TT_ERR_AWTRUNTIMENOTFOUND	5256
TT_ERR_AWTORACLEUIDPWDNOTINITED	5257
TT_ERR_CACHEALLOCSTMTFAILED	5800
TT_ERR_CACHEGETATTRFAILED	5801
TT_ERR_CACHEGETLOGSTATUS	5900
TT_ERR_CACHELOGTBLNOTFOUND	5901
TT_ERR_CACHEORACLENAMELEN	5902
TT_ERR_AREFRESHNOORADATA	5903
TT_ERR_AREFRESHNOADMINUID	5904
TT_ERR_AREFRESHNOADMINPWD	5905
TT_ERR_CACHENOARUSER	5907
TT_ERR_CACHENOARPWD	5908
TT_ERR_CACHENOADMINUSER	5909
TT_ERR_CACHEARLOGINBAD	5911
TT_ERR_CACHEADMINUSERNOTFOUND	5912
TT_ERR_TABLENOTSUPPORTED	5913
TT_ERR_CACHETBLNOTFOUND	5914
TT_ERR_BDBAROBJIDNOTFOUND	5917
TT_ERR_BDBARLOGTABLENOTFOUND	5918

TT_ERR_BDBARCGINVALID	5919
TT_ERR_BDBAROBJIDORLOGTBLNOTFOUND	5920
TT_ERR_ORACLEDOWN	5921
TT_ERR_MINSTOPTIMEOUTWARN	5922
TT_ERR_ORACLEDOWNREQUESTDEFERRED	5923
TT_ERR_AGENTORACLENOTINITIALIZED	5924
TT_ERR_MISMATCHAUTOREFTYPE	5931
TT_ERR_AUTOREFTYPEBADVALUE	5932
TT_ERR_VALIDATEORACLELOGIN	5935
TT_ERR_ORAAGENTUIDPWD	5936
TT_ERR_ORAAGENTNEEDUID	5937
TT_ERR_BDBUSERCOUNTNOTFOUNDWARN	5938
TT_ERR_BDBTBLNAMEUNKNOWN	5939
TT_ERR_CACHEARUIDVERIFY	5940
TT_ERR_CACHEARUIDCHANGED	5941
TT_ERR_ORAAGENTUIDPWDMATCH	5942
TT_ERR_DIFFUID	5943
TT_ERR_DIFFPWD	5944
TT_ERR_TRUNCNOTLOCKED	5945
TT_ERR_KEEPCGMUST	5946
TT_ERR_DUPLUIDPWDREQUIRED	5947
TT_ERR_AREFRESHFATAL	5990
TT_ERR_AREFRESHRETRY	5991
TT_ERR_CACHECLEANUPLOCKTIMEOUT	5992
TT_ERR_FTUSERCOUNTNOTCOMPATIBLE	5994
TT_ERR_BADLOCKINGLEVEL	6000
TT_ERR_CONDLOCKCONFLICT	6001

TT_ERR_DEADLOCKVICTIM	6002
TT_ERR_TIMEOUTVICTIM	6003
TT_ERR_BADISOLEVEL	6004
TT_ERR_LOCKSPACE	6005
TT_ERR_BADLOCKWAITVALUE	6006
TT_ERR_ISOLEVELCONFLICT	6009
TT_ERR_LOCKDBINVALID	6010
TT_ERR_LOCKSLEEP	6011
TT_ERR_BADCHECKOVERRIDE	6013
TT_ERR_BADTUPLISTMAINT	6014
TT_ERR_CURISEMPTY	6101
TT_ERR_CURNOTUPDATABLE	6102
TT_ERR_CURHASNOCURRENT	6103
TT_ERR_COLNOTUPDATABLE	6104
TT_ERR_OPENIXCURCONFLICT	6105
TT_ERR_OPENTBLCURCONFLICT	6106
TT_ERR_BADCURSOR	6107
TT_ERR_CURSTATEUNPOSITIONED	6108
TT_ERR_CURUPDATECONFLICT	6109
TT_ERR_CONFLICTWITHINDEX	6110
TT_ERR_SQLQUERYTIMEOUT	6111
TT_ERR_CONNECTSIZESMALL	6200
TT_ERR_SIZEOVERFLOW	6203
TT_ERR_HPISFULL	6206
TT_ERR_MEMORYLOCKNOTSUPPORTED	6212
TT_ERR_CANNOTSHMGETUSERSHMKEY	6213
TT_ERR_CANNOTSTATUSERSHMKEY	6214

TT_ERR_USERSHMKEYTOOSMALL	6215
TT_ERR_USERSHMKEYNATTCHNOTZERO	6216
TT_ERR_PERMSPACEEXHAUSTED	6220
TT_ERR_TEMPSPACEEXHAUSTED	6221
TT_ERR_PERMTHRESHOLDEXCEEDED	6222
TT_ERR_TEMPTHRESHOLDEXCEEDED	6223
TT_ERR_FIRSTCONNATTRIB	6226
TT_ERR_LOGFILESZADJ	6227
TT_ERR_INVALIDCONNATTRIBVALUE	6228
TT_ERR_TOOMANYTABLES	7000
TT_ERR_USERAUTHENTICATION	7001
TT_ERR_NOCURRENTUSER	7002
TT_ERR_FLTNAN	7003
TT_ERR_DBLNAN	7004
TT_ERR_PROCEDUREDOESNOTEXIST	7005
TT_ERR_ARGBAD	7006
TT_ERR_ARGREQUIRED	7007
TT_ERR_READONLY	7008
TT_ERR_TOOMANYPARAMS	7009
TT_ERR_PROCFAILED	7011
TT_ERR_SYSTEMTBLUPDATE	7012
TT_ERR_SCANNING	7013
TT_ERR_ARGTOOBIG	7014
TT_ERR_SYSTEMTBLIX	7015
TT_ERR_BADOPTFLAG	7017
TT_ERR_SYSTEMNAME	7018
TT_ERR_IXNAMEMATCH	7019

TT_ERR_NOTDURABLE	7020
TT_ERR_GROWRECOVERY	7022
TT_ERR_REBUILDREDO	7023
TT_ERR_BADSAMPLESTR	7024
TT_ERR_RESERVED	7025
TT_ERR_SYSTEMTABLEALTER	7026
TT_ERR_IXKEYEVAL	7027
TT_ERR_TABLENAMEBLANK	7028
TT_ERR_INDEXNAMEBLANK	7029
TT_ERR_TRACECOMPONENT	7050
TT_ERR_TRACEGENERIC	7051
TT_ERR_THREADSUPPORTFAILED	7052
TT_ERR_NOTHREADSUPPORT	7053
TT_ERR_NOUNIQUEINDEX	8000
TT_ERR_INVALIDLSNREAD	8001
TT_ERR_TBLISREPLICATED	8002
TT_ERR_OLDREPUPDATE	8003
TT_ERR_REPBADTSCOLUMN	8004
TT_ERR_REPBADTSUPDATERULE	8005
TT_ERR_BADREPUPDATE	8006
TT_ERR_BUFFSIZE	8009
TT_ERR_REPSEGATTACH	8010
TT_ERR_REPSEGACTIVE	8011
TT_ERR_NOSUCHTBLID	8012
TT_ERR_ILLEGALVERSION	8013
TT_ERR_REPNOCONFIG	8016
TT_ERR_REPDIFFCOLUMNCOUNT	8017

TT_ERR_REPDIFFPRIMCOLS	8018
TT_ERR_REPDIFFCOLTYPES	8019
TT_ERR_REPBADTUPLE	8020
TT_ERR_ILLEGALTBLID	8021
TT_ERR_NOSHARE	8022
TT_ERR_ROWMISMATCH	8023
TT_ERR_BADXLARECORD	8024
TT_ERR_REPLICATIONINVALID	8025
TT_ERR_REPNETWORKLOAD	8027
TT_ERR_REPNETWORKTIMEDOUT	8028
TT_ERR_XLABOOKMARKUSED	8029
TT_ERR_XLALSNBAD	8031
TT_ERR_STALEPOSTHANDLE	8032
TT_ERR_INVALIDPOSTHANDLE	8033
TT_ERR_XLANOSQL	8034
TT_ERR_XLANOLOGGING	8035
TT_ERR_XLAPARAMETER	8036
TT_ERR_XLATABLEDIFF	8037
TT_ERR_XLATABLESYSTEM	8038
TT_ERR_DROPREPTS	8039
TT_ERR_REPLCTNARRFULL	8040
TT_ERR_BADREPLCTNSLOT	8041
TT_ERR_REPLCTNSLOTFREE	8042
TT_ERR_REPMATERIALIZEDVIEW	8043
TT_ERR_REPDUPLICATENEEDED	8044
TT_ERR_XLATUPLEMISMATCH	8046
TT_ERR_XLADEDICATEDCONNECTION	8047

TT_ERR_DISKLESSTEMP	8048
TT_ERR_CTNWRAPAROUND	8049
TT_ERR_REPALTERTABLE	8050
TT_ERR_REPALTERTWOSAFE	8051
TT_ERR_OCTWOSAFE	8052
TT_ERR_REPSYNCNOSTORE	8053
TT_ERR_REPSYNCMUSTORE	8054
TT_ERR_REPVARMAX	8055
TT_ERR_REPTEMPORARYTABLE	8056
TT_ERR_REPAUTOCOMMIT	8099
TT_ERR_REPINVALIDVAL	8100
TT_ERR_REPDUPELEMENT	8101
TT_ERR_REPDUPSTORE	8102
TT_ERR_REPDUPSTOREATTRIBUTE	8103
TT_ERR_REPELEMENTDROPPED	8104
TT_ERR_REPNOROLE	8105
TT_ERR_REPOWNERSUBSCRIBER	8106
TT_ERR_REPDSTBLELEMENTS	8107
TT_ERR_REPNODSNONDURTRANSMIT	8108
TT_ERR_REPNOSUBSCRIBERELEMENT	8109
TT_ERR_REPCATCHUPREQ	8110
TT_ERR_REPCATCHUPPRG	8111
TT_ERR_REPCATCHUPOP	8112
TT_ERR_REPDDL	8113
TT_ERR_REPCATCHUPRESET	8114
TT_ERR_REPCANTALTER	8115
TT_ERR_REPPORTOUTOFRANGE	8116

TT_ERR_DSELEMENTNOTUNIQUE	8117
TT_ERR_COLTOOLONGFORREP	8118
TT_ERR_REPIDENTIFIEREQUAL	8119
TT_ERR_DURABLEXMITNOTALLOWED	8120
TT_ERR_LOCALHOSTINVALID	8121
TT_ERR_REPINVALIDSCHEMENAME	8122
TT_ERR_REPNOACTIVESTANDBY	8123
TT_ERR_REPACTIVESTANDBYANDSUB	8124
TT_ERR_REPCANTDROPMASTER	8125
TT_ERR_REPCANTCREATESCHEMEWITHAS	8126
TT_ERR_REPCANTASWITHOTHERSCHEME	8127
TT_ERR_REPASSTOREALREADYINSCHEME	8128
TT_ERR_REPASCANTSETSTOREATTR	8129
TT_ERR_REPASMUSTCREATEONASNODE	8130
TT_ERR_REPINCEXCALREADYINSCHEME	8131
TT_ERR_REPINCEXCNOTONTABLEELT	8132
TT_ERR_REPCANTINCEXCCGTABLE	8133
TT_ERR_REPSTORENOTMASTOFELT	8134
TT_ERR_REPEXCTABNOTINELT	8135
TT_ERR_REPEXCELTEMPTY	8136
TT_ERR_REPTABNOTINASSCHEME	8137
TT_ERR_REPEXCASEMPTY	8138
TT_ERR_REPBADFAILCALL	8139
TT_ERR_REPBADAS	8140
TT_ERR_REPLOCALNOACTIVE	8141
TT_ERR_REPHDRSTATFAIL	8142
TT_ERR_REPDUPACTIVE	8143

TT_ERR_REPSTOREDUPPERM	8144
TT_ERR_REPCANTALTERNOTACTIVE	8145
TT_ERR_REPNOTMOREONEINCEXC	8146
TT_ERR_CANTINCEXCFORONTABLEELT	8147
TT_ERR_REPHOSTNAMETOOLONG	8148
TT_ERR_REPDSNAMETOOLONG	8149
TT_ERR_REPDYNCHNGNOTALLOWED	8150
TT_ERR_REPACCESSVIOLATION	8151
TT_ERR_REPBADCOLTYPE	8152
TT_ERR_REPDUPOWNER	8153
TT_ERR_REPDUPREPLICATION	8154
TT_ERR_REPDUPSUBSCRIPTION	8155
TT_ERR_REPDUPTABLE	8156
TT_ERR_REPNOELEMENTS	8157
TT_ERR_REPNOLSN	8158
TT_ERR_REPNOPEERS	8159
TT_ERR_REPNOREPLICATION	8160
TT_ERR_REPNOSTORES	8161
TT_ERR_REPNOSUBSCRIPTIONS	8162
TT_ERR_REPNOTABLEORCOLDESCR	8163
TT_ERR_REPRELEASEMISMATCH	8164
TT_ERR_REPSCHEMACORRUPT	8165
TT_ERR_ROLLBACKNOTPOSSIBLE	8166
TT_ERR_SYSTABLEREPOPTIONSGET	8167
TT_ERR_SYSTABLEREPOPTIONSSET	8168
TT_ERR_REPDUPPATH	8169
TT_ERR_REPRETURNFAILED	8170

TT_ERR_REPSYSTEMTABLE	8171
TT_ERR_REPDUPSTORENAME	8172
TT_ERR_REPWITHXA	8173
TT_ERR_REPDUPTSATTRIBUTE	8174
TT_ERR_REPBADTSREPORTFILE	8175
TT_ERR_REPCONFLICTCHECKCONFLICT	8176
TT_ERR_REPSTOREROLE	8177
TT_ERR_REPBADPEERSTATE	8178
TT_ERR_REPBADDUPSTORE	8179
TT_ERR_REPPARTIALFULLDSCONFLICT	8180
TT_ERR_REPCREATEALTERINFULLDS	8181
TT_ERR_REPCONFLICTCHECKFULLDS	8182
TT_ERR_REPALTERSYSOWNEDELEMENT	8183
TT_ERR_REPCONFIGEMPTYFULLDS	8184
TT_ERR_REPRRLIMIT	8185
TT_ERR_REPRRNORESPONSE	8186
TT_ERR_REPRRNOTEXIST	8187
TT_ERR_REPRRNOTVALID	8188
TT_ERR_REPRRNOPEER	8189
TT_ERR_REPHOSTRESOLUTION	8190
TT_ERR_REPNOLOCALSTORE	8191
TT_ERR_REPFKCHNGNOTALLOWED	8192
TT_ERR_REPSCHEMANOTTWOSAFE	8193
TT_ERR_NOPROPTWOSAFE	8194
TT_ERR_NOBITWOSAFE	8195
TT_ERR_REPREMOTEERROR	8196
TT_ERR_REPTWOSAFECOMMIT	8197

TT_ERR_REPTSNOTEXIST	8198
TT_ERR_REPTSRRNOTEXIST	8199
TT_ERR_DROPALLCOLS	8200
TT_ERR_DROPPKEYCOL	8201
TT_ERR_DROPFKEYCOL	8202
TT_ERR_DROPINDEXEDCOLS	8203
TT_ERR_MAXTBLPTNS	8204
TT_ERR_REPBADSTORENAME	8205
TT_ERR_TABLEAMBIGUOUS	8210
TT_ERR_TABLENOTDEFINED	8211
TT_ERR_MULTIPLEPARENTFORTABLE	8212
TT_ERR_NOPK	8213
TT_ERR_ILLEGALSOURCE	8214
TT_ERR_INVALIDDURATION	8217
TT_ERR_ILLEGALATTRIBUTE	8220
TT_ERR_NOPARENTTABLE	8221
TT_ERR_MULTIPLEPARENTTABLE	8222
TT_ERR_INVALIDFK	8223
TT_ERR_CACHEGROUPEXISTS	8224
TT_ERR_TABLEREADONLY	8225
TT_ERR_COLUMNREADONLY	8226
TT_ERR_CACHEGROUPDOESNOTEXIST	8227
TT_ERR_USEDRCACHEGROUP	8228
TT_ERR_BADMEMORYTHRESHOLD	8229
TT_ERR_BADAGINGINTERVAL	8230
TT_ERR_CVUNSUPPORTED	8231
TT_ERR_PASSTHROUGHUNSUPPORTED	8232

TT_ERR_PASSTHROUGHBADVALUE	8233
TT_ERR_PASSTHROUGHCMDNOTUSABLE	8234
TT_ERR_WITHMISMATCH	8235
TT_ERR_INVALIDCOMMIT	8236
TT_ERR_CACHEWHERETOOBIG	8237
TT_ERR_MULTPROPAGATE	8239
TT_ERR_MULTREADONLY	8240
TT_ERR_NOAGING	8241
TT_ERR_EXPIXUNSUPPORTED	8242
TT_ERR_INVALIDARINTERVAL	8243
TT_ERR_ALTERAUTOREFRESH	8244
TT_ERR_NEEDAUTOREF	8247
TT_ERR_CGNOTAUTOREFRESHED	8248
TT_ERR_CGREFRESHERACTIVE	8249
TT_ERR_NOAGINGWAUTOREFRESH	8250
TT_ERR_BADNOTPROPAGATE	8252
TT_ERR_SAMEASCACHEGROUP	8253
TT_ERR_NOFLUSH	8254
TT_ERR_BADCACHEPROP	8255
TT_ERR_BADSUBQUERY	8256
TT_ERR_BADORACLEPARAM	8257
TT_ERR_REPCGDURATIONMISMATCH	8258
TT_ERR_REPCGWHEREMISMATCH	8259
TT_ERR_REPCGJOINMISMATCH	8260
TT_ERR_NOORACLEWITHXA	8261
TT_ERR_REPCGTYPE	8262
TT_ERR_REPCGAUTOREFRESH	8263

TT_ERR_ORACLETXFAILURE	8264
TT_ERR_NOTALLREADONLYORPROP	8265
TT_ERR_NEEDINCAUTOREFMODE	8266
TT_ERR_INVALIDWITHLIMIT	8267
TT_ERR_CACHETYPEWHERE	8268
TT_ERR_CACHETYPEAGING	8269
TT_ERR_CACHETYPETBLFLAG	8270
TT_ERR_CACHEMANUALOPTYPE	8271
TT_ERR_CVREQUIRED	8272
TT_ERR_REPCGAUTOREFSTATE	8273
TT_ERR_REPCGPROPAGATE	8274
TT_ERR_CACHEMANUALOPREF	8275
TT_ERR_ALTERAUTOREFLIMIT	8276
TT_ERR_REPCGAUTOREFRESHRCVR	8277
TT_ERR_REPCGBOTHAUTOREFSTATE	8278
TT_ERR_REPCGBOTHPROPAGATE	8279
TT_ERR_BIPROPWLOGMINER	8280
TT_ERR_WITHLIMIT	8286
TT_ERR_CACHEREFRESHAUTOREF	8287
TT_ERR_CACHELOADNOTEMPTY	8288
TT_ERR_CACHEREFSTATE	8289
TT_ERR_CACHEMUSTCOMMITN	8290
TT_ERR_CACHEMUSTNOTCOMMITN	8291
TT_ERR_UNKNOWNDAEMONERR	9990
TT_ERR_INVALIDATEINPROG	9991
TT_ERR_DAEMONFAILURESTRING	9992
TT_ERR_FINALCKPTINPROG	9993

TT_ERR_RECOVERYINPROG	9994
TT_ERR_FORCERLBKINPROG	9995
TT_ERR_EXAMINEINPROG	9996
TT_ERR_SYSTEMQUIESCING	9998
TT_ERR_SUBDAEMONDIED	9999
TT_ERR_DISKLESSMODE	10001
TT_ERR_UNKNOWNDATASTORE	10002
TT_ERR_UNEXPECTEDCKPTFILE	10003
TT_ERR_DAEMONPROTOCOL	10004
TT_ERR_DATASTORENOTINFLUX	10005
TT_ERR_POLICYRQSTINPROGRESS	10006
TT_ERR_FAILEDWAITNOTINFLUX	10007
TT_ERR_DATASTOREDELETED	10008
TT_ERR_RAMLOADIGNORED	10009
TT_ERR_RAMUNLOADIGNORED	10010
TT_ERR_DSNOTMANAGED	10011
TT_ERR_REPSTARTIGNORED	10012
TT_ERR_REPSTARTFAILED	10013
TT_ERR_REPSPAWNFAILED	10014
TT_ERR_REPSTOPFAILED	10015
TT_ERR_REPSTOPIGNORED	10016
TT_ERR_ORACLEIDNOTSET	10017
TT_ERR_ORACLEHOMENOTSET	10018
TT_ERR_ORACLEHOMELONG	10019
TT_ERR_ORASTOPFAILED	10020
TT_ERR_ORASTOPIGNORED	10021
TT_ERR_ORACLEHOMESETFAIL	10022

TT_ERR_LIBPATHSETFAIL	10023
TT_ERR_ORASTARTFAILED	10024
TT_ERR_ORASPAWNFAILED	10025
TT_ERR_XA	11000
TT_ERR_XAASYNC	11002
TT_ERR_XARMERR	11003
TT_ERR_XANOTA	11004
TT_ERR_XAINVAL	11005
TT_ERR_XAPROTO	11006
TT_ERR_XARMFAIL	11007
TT_ERR_XADUPID	11008
TT_ERR_XAOUTSIDE	11009
TT_ERR_XARDONLY	11013
TT_ERR_XARETRY	11014
TT_ERR_XAHEURRB	11016
TT_ERR_XAHEURCOM	11017
TT_ERR_XARBROLLBACK	11020
TT_ERR_XAAUTOCOMMIT	11030
TT_ERR_XASQLTRANSACT	11031
TT_ERR_XAEXCLACCESS	11032
TT_ERR_XANOLOGGING	11033
TT_ERR_XAXACTSAWAITINGRESOLUTION	11035
TT_ERR_XAWITHREP	11036
TT_ERR_XAWITHAGENT	11037
TT_ERR_XABEGINNOTFOUND	11038
TT_ERR_XARECONSTRUCT	11039
TT_ERR_UTILINVALIDDSN	12001

TT_ERR_UTILINIFILENOTFOUND	12002
TT_ERR_UTILINICANTOPENFILE	12003
TT_ERR_UTILINISYNTAX	12004
TT_ERR_UTILENVVARNOTSET	12005
TT_ERR_UTILINVALIDATTR	12006
TT_ERR_UTILINICANTBEEMPTY	12007
TT_ERR_UTILINIVALUETOOLONG	12008
TT_ERR_UTILINIDSNNOTFOUND	12009
TT_ERR_UTILINVALIDBUFF	12010
TT_ERR_UTILVALUETRUNCATED	12011
TT_ERR_UTILMALLOCFAILED	12012
TT_ERR_UTILINVALIDDATASTORE	12013
TT_ERR_UTILCONTEXTGET	12014
TT_ERR_UTILCONTEXTSET	12015
TT_ERR_UTILINVALIDRAMGRACE	12016
TT_ERR_UTILALLOCENV	12017
TT_ERR_UTILALLOCCONNECT	12018
TT_ERR_UTILDATASTORELOC	12019
TT_ERR_UTILCONNECTFAILED	12020
TT_ERR_UTILDISCONNECTFAILED	12021
TT_ERR_UTILFREECONNECTFAILED	12022
TT_ERR_UTILFREEENVFAILED	12023
TT_ERR_UTILALREADYLOADED	12024
TT_ERR_UTILALREADYUNLOADED	12025
TT_ERR_UTILALREADYRUNNING	12026
TT_ERR_UTILALREADYSTOPPED	12027
TT_ERR_UTILUNKNOWNDATASTORE	12028

TT_ERR_UTILEXCLUSIVECONNECTION	12029
TT_ERR_UTILDATASTOREINUSE	12030
TT_ERR_UTILEXCLACCESS	12031
TT_ERR_UTILINVALIDRAMPOL	12032
TT_ERR_UTILEXCLRAMPOL	12033
TT_ERR_UTILTEMPRAMPOL	12034
TT_ERR_UTILINITWINSOCK	12035
TT_ERR_UTILWINSOCKNOTFOUND	12036
TT_ERR_UTILCLOSEWINSOCK	12037
TT_ERR_REMOTEDAEMONCONNFAIL	12038
TT_ERR_REMOTEREPPORTFAIL	12039
TT_ERR_REMOTEREPCONNFAIL	12040
TT_ERR_UTILHOSTNOTFOUND	12041
TT_ERR_UTILXACTBEGIN	12042
TT_ERR_UTILTEMPFILE	12043
TT_ERR_UTILSENDFAIL	12044
TT_ERR_UTILFILECREATE	12045
TT_ERR_UTILRECVFAIL	12046
TT_ERR_UTILWRONGRESP	12047
TT_ERR_UTILSRCBACKUP	12048
TT_ERR_UTILFILEWRITE	12049
TT_ERR_UTILSELFXACTCOMMIT	12050
TT_ERR_UTILHOSTNAMETOOLONG	12051
TT_ERR_UTILLONGSCHEMENAME	12052
TT_ERR_UTILLONGSCHEMEOWNER	12053
TT_ERR_UTILGETUSERFAIL	12054
TT_ERR_UTILREPNOCHANGE	12055

TT_ERR_UTILSIZEMISMATCH	12056
TT_ERR_UTILTEMPDSCREATE	12057
TT_ERR_UTILREPBACKCTN	12058
TT_ERR_UTILDEADLOCK	12059
TT_ERR_UTILINVALIDREMOTEHOST	12060
TT_ERR_UTILINVALIDLOCALHOST	12061
TT_ERR_UTILREPWRONGVERSION	12062
TT_ERR_UTILFILEREAD	12063
TT_ERR_UTILREPUPDEXECFAIL	12064
TT_ERR_UTILMULTIREPVER	12065
TT_ERR_UTILCOMPILECMD	12066
TT_ERR_UTILOPENSTMT	12067
TT_ERR_UTILNEXTTUP	12068
TT_ERR_UTILFORMATHANDLE	12069
TT_ERR_UTILCOLINFOFAIL	12070
TT_ERR_UTILCOLVALFAIL	12071
TT_ERR_UTILMULTISTOREID	12072
TT_ERR_UTILSTOREMINE	12073
TT_ERR_UTILHOSTMINE	12074
TT_ERR_UTILREPVEROLD	12075
TT_ERR_UTILREPVERNEW	12076
TT_ERR_UTILHOSTNAME	12077
TT_ERR_UTILRESETLOCALSTORE	12078
TT_ERR_UTILSETLOCALSTORE	12079
TT_ERR_UTILNORECEIVER	12080
TT_ERR_UTILINVALIDCOLTYPE	12081
TT_ERR_UTILRESETRCVCTN	12082

TT_ERR_UTILNOSRCID	12083
TT_ERR_UTILGETHOSTNAME	12084
TT_ERR_UTILNOTBLFOUND	12085
TT_ERR_UTILNOTBLINFO	12086
TT_ERR_UTILNOTBLFORMAT	12087
TT_ERR_UTILMULTISRCID	12088
TT_ERR_UTILNOTBLCOLINFO	12089
TT_ERR_UTILCOLNOTYPE	12090
TT_ERR_UTILVER2REPTBL	12091
TT_ERR_UTILSCHEMANOTEMPTY	12092
TT_ERR_UTILTBLSNOCOUNT	12093
TT_ERR_UTILTBLSNOCOUNTINFO	12094
TT_ERR_UTILUPDATEOBJID	12095
TT_ERR_UTILSETOBJID	12096
TT_ERR_UTILPEERNOCOLVAL	12097
TT_ERR_UTILPEERWRITELSN	12098
TT_ERR_UTILMULTIREPSCHM	12099
TT_ERR_UTILCREATEREPTDLDEF	12100
TT_ERR_UTILREPVERXACTBEGIN	12101
TT_ERR_UTILREPVERXACTCOMMIT	12102
TT_ERR_UTILREPELEMSCANCOLFAIL	12103
TT_ERR_UTILREPWRITELSN	12104
TT_ERR_UTILUPDATELSN	12105
TT_ERR_UTILSETREPLSN	12106
TT_ERR_UTILDELETETBLDEF	12107
TT_ERR_INVALIDPOLICY	12108
TT_ERR_CONNSTRSYNTAX	12109

TT_ERR_UTILNODSNDRIVER	12110
TT_ERR_UTILSIGNALRCVD	12111
TT_ERR_UTILALLOCSTMT	12112
TT_ERR_REPAGNTPORTFAIL	12113
TT_ERR_UTILBACKUPVER	12114
TT_ERR_REPAGENTCONNECTFAIL	12115
TT_ERR_UTILRESTDBCREATE	12116
TT_ERR_UTILRESTDBDISCONN	12117
TT_ERR_REPAGENTSENDFAIL	12118
TT_ERR_UTILDESTFILEOPEN	12119
TT_ERR_UTILSTRMINSUFCNTRD	12120
TT_ERR_UTILSTRMFILEHDRBAD	12125
TT_ERR_UTILDSNAMECREATE	12132
TT_ERR_UTILDBFILEEXISTS	12133
TT_ERR_UTILLOGFILEEXISTS	12134
TT_ERR_UTILDBFILEDELETE	12135
TT_ERR_UTILLOGFILEDELETE	12136
TT_ERR_UTILINVALIDCONNSTR	12148
TT_ERR_UTILRCVCTNXACTBEGIN	12150
TT_ERR_UTILRCVCTNXACTCOMMIT	12151
TT_ERR_UTILREPTSSETFAIL	12152
TT_ERR_UTILDSOBJIDFAIL	12153
TT_ERR_UTILREPTBLIDFAIL	12154
TT_ERR_UTILREPROWIDFAIL	12155
TT_ERR_UTILREPSECONDPASS	12156
TT_ERR_UTILREPTBLUPD	12157
TT_ERR_UTILREPTBLUPDATEFAIL	12158

TT_ERR_UTILNULLPOINTER	12159
TT_ERR_UTILTEMPBACKUP	12161
TT_ERR_UTILTEMPRAMGRACE	12164
TT_ERR_UTILBACKUPSTART	12166
TT_ERR_UTILDAEMONINITCONN	12184
TT_ERR_UTILNOSTOREID	12185
TT_ERR_UTILINISTATFAIL	12186
TT_ERR_UTILINIREADFAIL	12187
TT_ERR_UTILINICLOSEFAIL	12188
TT_ERR_UTILINISCANSTOP	12189
TT_ERR_UTILINIATTRLONG	12190
TT_ERR_INICONNSTRLEN	12191
TT_ERR_UTILBACKUPTYPE	12192
TT_ERR_UTILBACKUPNODIR	12193
TT_ERR_UTILBACKUPDBINFO	12194
TT_ERR_UTILBACKUPBASENAMEBAD	12195
TT_ERR_UTILBACKUPNOSTAFILE	12196
TT_ERR_UTILBACKUPSTAMISMATCH	12197
TT_ERR_UTILBACKUPFILESMISSING	12198
TT_ERR_UTILBACKUPDIRREAD	12199
TT_ERR_UTILBACKUPSTAFILEOPEN	12200
TT_ERR_UTILBACKUPSTAFILEREAD	12201
TT_ERR_UTILBACKUPSTAFILEWRITE	12202
TT_ERR_UTILBACKUPSTAFILECLOSE	12203
TT_ERR_UTILBACKUPSTAFILEBAD	12204
TT_ERR_UTILBACKUPINCOMPLETE	12205
TT_ERR_UTILBACKUPRECBAD	12206

TT_ERR_UTILBACKUPRECWRITE	12207
TT_ERR_UTILBACKUPFILEINFO	12208
TT_ERR_UTILBACKUPFILEOPEN	12209
TT_ERR_UTILBACKUPFILEREAD	12210
TT_ERR_UTILBACKUPFILESHORT	12211
TT_ERR_UTILBACKUPFILEWRITE	12212
TT_ERR_UTILBACKUPFILECLOSE	12213
TT_ERR_UTILBACKUPFILERENAME	12214
TT_ERR_UTILBACKUPTERMINATED	12215
TT_ERR_UTILRESTORETYPE	12216
TT_ERR_UTILTEMPRESTORE	12217
TT_ERR_SERVERPORTFAIL	12218
TT_ERR_SERVERCONNECTFAIL	12219
TT_ERR_SERVERSENDFAIL	12220
TT_ERR_UTILBACKUPPLAT	12221
TT_ERR_UTILBACKUPOLD	12222
TT_ERR_UTILSERVERNOCONN	12223
TT_ERR_UTILINVALIDHDBC	12224
TT_ERR_UTILCSERROR	12225
TT_ERR_UTILCSNOTIMPL	12226
TT_ERR_UTILDELXLA	12227
TT_ERR_UTILNOTLOADED	12228
TT_ERR_REPAGENTRECVFAIL	12229
TT_ERR_UTILINVALIDARGUMENT	12230
TT_ERR_LICFILEMISSING	13000
TT_ERR_LICFILEUNREADABLE	13001
TT_ERR_LICFILEINVALID	13002

TT_ERR_LICENSEEXPIRED	13003
TT_ERR_LICENSEMAXCONNSEXCEEDED	13004
TT_ERR_LICENSEMAXSIZEEXCEEDED	13005
TT_ERR_LICENSEMGRINTERNALERROR	13006
TT_ERR_DAEMONINTERNALERR	14000
TT_ERR_DAEMONOUTOFMEMORY	14001
TT_ERR_DAEMONPORTINUSE	14002
TT_ERR_DAEMONTHREADCREATE	14003
TT_ERR_DAEMONCREATEFAILURE	14004
TT_ERR_DAEMONCONNECTFAILURE	14005
TT_ERR_DAEMONDISCONNECTFAILURE	14006
TT_ERR_DAEMONDELETEFAILURE	14007
TT_ERR_DAEMONASSOCTABLEFULL	14008
TT_ERR_DAEMONSTARTUPFAILURE	14009
TT_ERR_DAEMONSPAWNFAILURE	14010
TT_ERR_DAEMONINVALIDATIONFAILURE	14011
TT_ERR_DAEMONSUBDAEMONFAILURE	14012
TT_ERR_DAEMONOUTOFSUBDAEMONS	14013
TT_ERR_DAEMONPOLICYFAILURE	14014
TT_ERR_DAEMONSETUIDUSER	14015
TT_ERR_AUTHINCORRECTPASSWD	15000
TT_ERR_AUTHNOPRIVILEGE	15001
TT_ERR_AUTHNOSUCHUSER	15002
TT_ERR_AUTHTOOMANYUSERS	15003
TT_ERR_AUTHUSEREXISTS	15004
TT_ERR_AUTHCANNOTDELETEADMIN	15006
TT_ERR_AUTHNOTENABLED	15007

TT_ERR_AUTHUSERNAMETOOLONG	15008
TT_ERR_AUTHPASSWDREQUIRED	15009
TT_ERR_AUTHCONVERTUSERTYPE	15010
TT_ERR_AUTHPASSWDTOOLONG	15011
TT_ERR_AUTHENCPASSWDLENGTH	15012
TT_ERR_AUTHCANNOTALTERADMIN	15013
TT_ERR_AUTHNOTINSTANCEADMIN	15014
TT_ERR_AUTHCANNOTMODPASSADMIN	15015
TT_ERR_AUTHINTERNALSYSTEMUSER	15016
TT_ERR_AUTHNOATTRPRIVILEGE	15017
TT_ERR_AUTHPASSWDILLEGAL	15018
TT_ERR_REPAGENTSTARTED	16001
TT_ERR_REPAGENTSTOPPING	16002
TT_ERR_REPNORESTART	16003
TT_ERR_REPDBCONNECTFAIL	16004
TT_ERR_REPDBDISCONNECTFAIL	16005
TT_ERR_REPDBCTXTCREATEFAIL	16006
TT_ERR_REPDBCTXTSETFAIL	16007
TT_ERR_REPDBINFOFAIL	16008
TT_ERR_REPDISKLESSSTATFAIL	16009
TT_ERR_REPDISKLESSUPDTFAIL	16010
TT_ERR_REPDUPNEEDED	16011
TT_ERR_REPDBINVALID	16012
TT_ERR_REPDISKLESSTEMP	16013
TT_ERR_REPSCHEMAVERMISMATCH	16014
TT_ERR_REPSCHEMAVERFAIL	16015
TT_ERR_REPERRFROMDAEMON	16016

TT_ERR_REPDAEMONCONNINIT	16017
TT_ERR_REPDAEMONCONNFAIL	16018
TT_ERR_REPDEADMANREADFAIL	16019
TT_ERR_REPDEADMANDATA	16020
TT_ERR_REPDAEMONGONE	16021
TT_ERR_REPCOMMANDRCVD	16022
TT_ERR_REPTHEADCREATEFAIL	16023
TT_ERR_REPTHREADCREATENOMEM	16024
TT_ERR_REPTHEADSTARTING	16025
TT_ERR_REPTHEADEXITING	16026
TT_ERR_REPALLTHEADSEXITED	16027
TT_ERR_REPTHREADFAILCNT	16028
TT_ERR_REPTHREADEXITWAIT	16029
TT_ERR_REPNOTPARTICIPANT	16030
TT_ERR_REPMALLOCFAIL	16031
TT_ERR_REPSYSTEMCALLFAIL	16032
TT_ERR_REPINDOUBTXACTWAIT	16033
TT_ERR_REPINVALIDDBGOPTS	16034
TT_ERR_REPDBGOPTSSET	16035
TT_ERR_REPXMTRTHREADEXIST	16036
TT_ERR_REPXACTNESTED	16037
TT_ERR_REPXACTBEGINFAIL	16038
TT_ERR_REPXACTCOMMITFAIL	16039
TT_ERR_REPXACTROLLBACKFAIL	16040
TT_ERR_REPHOLDLSNSETFAIL	16041
TT_ERR_REPXMTRAWAKEFAIL	16042
TT_ERR_REPLSNGETFAIL	16043

TT_ERR_REPMYIDGETFAIL	16044
TT_ERR_REPPEERIDGETFAIL	16045
TT_ERR_REPFORCELOGFAIL	16046
TT_ERR_REPLOGGINGMISMATCH	16047
TT_ERR_REPVERSIONMISMATCH	16048
TT_ERR_REPCTNUPDATEFAIL	16049
TT_ERR_REPHOLDLSNUPDTFAIL	16050
TT_ERR_REPSTATEUPDTFAIL	16051
TT_ERR_REPDBLOCKSETFAIL	16052
TT_ERR_REPTEMPFILEOPENFAIL	16053
TT_ERR_REPTEMPFILESEEKFAIL	16054
TT_ERR_REPTEMPFILEREADFAIL	16055
TT_ERR_REPBACKUPFAIL	16056
TT_ERR_REPNETWORKCKPTFAIL	16057
TT_ERR_REPTBLRENEWFAIL	16058
TT_ERR_REPREPLYRQSTFAIL	16059
TT_ERR_REPSOCKETREADFAIL	16060
TT_ERR_REPCOMPILECMDFAIL	16062
TT_ERR_REPEXECUTECMDFAIL	16063
TT_ERR_REPEXECCMDFAILINFUNC	16064
TT_ERR_REPCURSOROPENFAIL	16065
TT_ERR_REPCUROPENFAILINFUNC	16066
TT_ERR_REPCURSORNEXTFAIL	16067
TT_ERR_REPCURNEXTFAILINFUNC	16068
TT_ERR_REPNOROWFOUND	16069
TT_ERR_REPCMDFMTARGFAIL	16070
TT_ERR_REPCMDFMTRESFAIL	16071

TT_ERR_REPCMDDELETEFAIL	16072
TT_ERR_REPCTNSLOTALLOCFAIL	16073
TT_ERR_REPTBLNOTFOUND	16074
TT_ERR_REPTBLLOOKUPFAIL	16075
TT_ERR_REPTBLVERSIONFAIL	16076
TT_ERR_REPTBLFORMATFAIL	16077
TT_ERR_REPINVALIDTBLDEF	16078
TT_ERR_REPTBLCOLINFOFAIL	16079
TT_ERR_REPROWLOOKUPACTFAIL	16080
TT_ERR_REPROWUPDATEFAIL	16081
TT_ERR_REPROWUPDACTFAIL	16082
TT_ERR_REPROWDELACTFAIL	16083
TT_ERR_REPROWINSACTFAIL	16084
TT_ERR_REPEMPTYTBLFAIL	16085
TT_ERR_REPPEERUPDTCNTWRONG	16086
TT_ERR_REPDUPOPTREADFAIL	16087
TT_ERR_REPUNIQCONSTRAINTFAIL	16088
TT_ERR_REPFKEYCONSTRAINTFAIL	16089
TT_ERR_REPCONSTRAINTCHKFAIL	16090
TT_ERR_REPTBLCONSTRAINTFAIL	16091
TT_ERR_REPINVALIDLOGRECTYPE	16092
TT_ERR_REPINVALIDCOLTYPE	16093
TT_ERR_REPEXECSQLFAIL	16094
TT_ERR_REPINDXMATCHFAIL	16095
TT_ERR_REPINDXINFOFAIL	16096
TT_ERR_REPTBLPTNWIDTHFAIL	16097
TT_ERR_REPINVALIDELEMCNT	16098

TT_ERR_REPINVALIDMASTERCNT	16099
TT_ERR_REPMULTIHOSTPEER	16100
TT_ERR_REPRETURNSVCMISMATCH	16101
TT_ERR_REPRETBYRQSTMISMATCH	16102
TT_ERR_REPCGLOOKUPFAIL	16103
TT_ERR_REPCGLENMISMATCH	16104
TT_ERR_REPCGINCOMPATIBLE	16105
TT_ERR_REPCVTBLUNSUBSCRIBED	16106
TT_ERR_REPCVCHECKFAIL	16107
TT_ERR_REPLOCALINFOFAIL	16108
TT_ERR_REPSENDLSNUPDTFAIL	16109
TT_ERR_REPRRSENDLSNUPDTFAIL	16110
TT_ERR_REPSENDLSNINITFAIL	16111
TT_ERR_REPMETAINITFAIL	16112
TT_ERR_REPGETIPADDRFAIL	16113
TT_ERR_REPIPCONNECTINFO	16114
TT_ERR_REPIPCONNECTEDINFO	16115
TT_ERR_REPADDCOLDEFINFO	16116
TT_ERR_REPDROPCOLDEFINFO	16117
TT_ERR_REPGETREPHOLDFAIL	16118
TT_ERR_REPSETLOGBUFFPADFAIL	16119
TT_ERR_REPSTATISTICSUPDTFAIL	16120
TT_ERR_REPFLUSHQUEUEFAIL	16121
TT_ERR_REPHEARTBEATSENDFAIL	16122
TT_ERR_REPXACTINQUEUE	16123
TT_ERR_REPTBLSUBSFAIL	16124
TT_ERR_REPEXCEEDINLINEMAX	16125

TT_ERR_REPLOGREADINITFAIL	16126
TT_ERR_REPLOGREADFAIL	16127
TT_ERR_REPLOGSEEKFAIL	16128
TT_ERR_REPLOGREADSTARTLSN	16129
TT_ERR_REPNOOLDVALINUPDT	16130
TT_ERR_REPLOGOFFSETWRONG	16131
TT_ERR_REPLOGBUFFADDRSZWRONG	16132
TT_ERR_REPPEERCNFIGCHANGED	16133
TT_ERR_REPWRONGPACKETTYPE	16134
TT_ERR_REPCOLINFOFAIL	16135
TT_ERR_REPCOLINFOSEND	16136
TT_ERR_REPXMTRSIGNONFAIL	16137
TT_ERR_REPNOCATCHUPFORPROP	16138
TT_ERR_REPADDTBLFAIL	16139
TT_ERR_REPSYNCPOSTPROCFAIL	16140
TT_ERR_REPSYNCSEMAINITFAIL	16141
TT_ERR_REPNOPEERFOUND	16142
TT_ERR_REPTIMEOUTINFOFAIL	16143
TT_ERR_REPRRBUFFALLOCFAIL	16144
TT_ERR_REPRRSTATEUPDTFAIL	16145
TT_ERR_REPDSKLSSCTNXCHNGFAIL	16146
TT_ERR_REPDSKLSSXMITBYTEFAIL	16147
TT_ERR_REPSCHEMACHANGEDINFO	16148
TT_ERR_REPXMTRFAILEDSTATE	16149
TT_ERR_REPFAILPKTFAIL	16150
TT_ERR_REPLOGBUFFTOOSMALL	16151
TT_ERR_REPISFAILEDFAIL	16152

TT_ERR_REPRRBUFFINITFAIL	16153
TT_ERR_REPISCHANGEDFAIL	16154
TT_ERR_REPCNFGACCESSTEMPFAIL	16155
TT_ERR_REPCNFGACCESSFAIL	16156
TT_ERR_REPRCVREXITWAIT	16157
TT_ERR_REPPEERLISTFAIL	16158
TT_ERR_REPSTARTXMTRINFO	16159
TT_ERR_REPLOGFLUSHFAIL	16160
TT_ERR_REPRCVRSOCKETINFO	16161
TT_ERR_REPSETSOCKOPTFAIL	16162
TT_ERR_REPRCVREXITING	16163
TT_ERR_REPFAILPKTRCVD	16164
TT_ERR_REPBACKUPOPFAIL	16165
TT_ERR_REPMEMCOPYOPFAIL	16166
TT_ERR_REPCATCHUPFAIL	16167
TT_ERR_REPCATCHUPSTARTED	16168
TT_ERR_REPCATCHUPSTOPPING	16169
TT_ERR_REPCATCHUPSTOPPED	16170
TT_ERR_REPPEERLENINVALID	16171
TT_ERR_REPWRONGPEERRCVD	16172
TT_ERR_REPPEERUNKNOWN	16173
TT_ERR_REPPEERAMBIGUOUS	16174
TT_ERR_REPPEERUSEDEFAULT	16175
TT_ERR_REPPEERALREADYSIGNEDON	16176
TT_ERR_REPPEERLOOKUPFAIL	16177
TT_ERR_REPPEERNOTINPROGRESS	16178
TT_ERR_REPPEERNOTDEFINED	16179

TT_ERR_REPPEERSTOREIDSAME	16180
TT_ERR_REPRESERVESLOTFAIL	16181
TT_ERR_REPCTNSLOTGETFAIL	16182
TT_ERR_REPSTATUSSETFAIL	16183
TT_ERR_REPPROPSTATUSFAIL	16184
TT_ERR_REPRCVRCNFIGCHANGEDINFO	16185
TT_ERR_REPRCVRVERMISMATCH	16186
TT_ERR_REPXACTFAILRESULT	16187
TT_ERR_REPTRANCOMMITFAIL	16188
TT_ERR_REPRCVRNOTIDLE	16189
TT_ERR_REPPROPLOOPPUSHFAIL	16190
TT_ERR_REPREPLACETBLDEF	16191
TT_ERR_REPSKIPADDTBLDEF	16192
TT_ERR_REPADDTBLDEFINFO	16193
TT_ERR_REPRETSRVCCONFLICT	16194
TT_ERR_REPELEMOWNERFAIL	16195
TT_ERR_REPELEMOWNERCONFLICT	16196
TT_ERR_REPADDELEMFAIL	16197
TT_ERR_REPTBLDEFMISMATCH	16198
TT_ERR_REPTBLDEFCOLMISMATCH	16199
TT_ERR_REPTBLDEFTSCOLMISMATCH	16200
TT_ERR_REPTBLDEFTSACTMISMATCH	16201
TT_ERR_REPNOCACHEGROUP	16202
TT_ERR_REPTBLCOMPPASSINFO	16203
TT_ERR_REPTBLMARKEDINVALID	16204
TT_ERR_REPCGDEFSKIP	16205
TT_ERR_REPCGDEFDIFFERENT	16206

TT_ERR_REPTBLIDMISMATCH	16207
TT_ERR_REPREPORTOPENFAIL	16208
TT_ERR_REPAPPLYLOGRECFAIL	16209
TT_ERR_REPKEYLENEXCEEDED	16210
TT_ERR_REPVCFINDFAIL	16211
TT_ERR_REPSETCOLFAIL	16212
TT_ERR_REPROWINSERTFAIL	16213
TT_ERR_REPROWDELETEFAIL	16214
TT_ERR_REPROWUPDTFAIL	16215
TT_ERR_REPTBLCOLNUMFAIL	16216
TT_ERR_REPCONNAWAKENED	16217
TT_ERR_REPCONNAWAKENFAIL	16218
TT_ERR_REPCOMPRESSINFOFAIL	16219
TT_ERR_REPRELEASESUNSUPPORTED	16220
TT_ERR_REPTBLNAMENOTFOUND	16221
TT_ERR_REPGETTIMEOFDAY	16222
TT_ERR_REPHIGHLATENCY	16223
TT_ERR_REPHIGHSKEW	16224
TT_ERR_REPSCHEMEDIFF	16225
TT_ERR_REPSCHEMEREAD	16226
TT_ERR_REPSTANDBYAHEAD	16227
TT_ERR_REPSCHEMESTDIFF	16228
TT_ERR_REPSCHEMESUBDIFF	16229
TT_ERR_REPNOTSTANDBY	16230
TT_ERR_REPDUPLICATENOTCOMPLETE	16231
TT_ERR_REPPEERHIGHERTS	16232
TT_ERR_REPPEERAHEAD	16233

TT_ERR_REPPEERFAILMISMATCH	16234
TT_ERR_REPSTANDBYUPDATEFAIL	16235
TT_ERR_REPSUBUPDATEFAIL	16236
TT_ERR_REPINCEXCOBJNOTUSED	17000
TT_ERR_REPDUPEIDENTS	17001
TT_ERR_REPEXCLUDEDALL	17002
TT_ERR_DEPRECATEDFEATURE	20000
TT_ERR_RECOVERYNEEDED	20100
TT_ERR_PARAMOPTIONSWITHRESULTS	30103

8

JMS/XLA Reference

This chapter provides reference information for the JMS/XLA API.

The topics in this section are:

XLA MapMessage contents

A **`MapMessage`** contains a set of typed name/value pairs that correspond to the fields in an XLA update header, which is published as the C structure `ttXlaUpdateDesc_t`. The fields contained in a `MapMessage` depend on what type of update it is.

Update type

Each `MapMessage` returned by the JMS/XLA API contains at least one name/value pair called `__TYPE` (with 2 underscores) that identifies the type of update described in the message as an integer value. The types are specified as integer values. As a convenience, you can use the constants defined in `com.timesten.dataserver.jmsxla.XlaConstants` to compare against the integer types. The supported types are shown in Table 8.1.

Table 8.1 Update Types

CREATE_TABLE	DROP_TABLE
CREATE_INDEX	DROP_INDEX
ADD_COLUMNS	DROP_COLUMNS
INSERT	UPDATE
DELETE	COMMIT_ONLY

CREATE_VIEW	DROP_VIEW
CREATE_SEQ	DROP_SEQ
TRUNCATE	

XLA flags

For all update types, the `MapMessage` contains name/value pairs that indicate:

- If this is the first record of a transaction.
- If this is the last record of a transaction.
- If the update was performed by replication.
- Which table was updated.
- The owner of the updated table.

The name/value pairs that contain these XLA Flags are described in Table 8.2. Each name is preceded by two underscores.

Table 8.2 XLA Flags

Name	Description	Corresponding ttXlaTblDesc_t Field
__COMMIT	Indicates that this is the last record in a transaction and that a commit was performed after this operation. Only included in the `MapMessage` if UPDCOMMIT is on.	TT_UPDCOMMIT
__FIRST	Indicates that this is the first record in a new transaction. Only included in the `MapMessage` if UPDFIRST is on.	TT_UPDFIRST

Table 8.2 XLA Flags

Name	Description	Corresponding ttXlaTblDesc_t Field
__REPL	Indicates that this change was applied to the database via replication. Only included in the MapMessage if UPDREPL is on.	TT_UPDREPL
__UPDCOLS	Only used for UPDATETUP records, this flag indicates that the XLA update descriptor contains a list of columns that were actually modified by the operation. Specified as a String that contains a semicolon delimited list of column names. Only included in the MapMessage if UPDCOLS is on.	TT_UPDCOLS

DML event data formats

Many Data Manipulation Language (DML) operations generate XLA updates that can be monitored by XLA event handlers. This section describes the contents of the MapMessage objects that are generated for these operations.

Table data

For INSERT, UPDATE and DELETE operations MapMessages contain two name/value pairs called __TBLOWNER and __TBLNAME. These fields describe the name and owner of the table that is being updated. For example, for a table owned by user SCOTT, called EMPLOYEES, which could be referred to in a SQL statement as SCOTT.EMPLOYEES, MapMessages related to this table contain a field called __TBLOWNER whose value is the string SCOTT, and a field called __TBLNAME whose value is the string EMPLOYEES.

Row data

For INSERT and DELETE operations, a complete image of the inserted or deleted row is included in the message and all column values are available.

For UPDATE operations, the complete before and after image of the row is available, as well as a list of column numbers indicating which columns were actually modified. You access the column values using the names of the columns. The column names in the "before" image all begin with a single underscore. For example, `<columnname>` contains the new value and `_<columnname>` contains the old value.

If the value of a column is NULL, it is omitted from the column list. The `__NULLS` name/value pair contains a semicolon-delimited list of the columns that contain NULL values.

Context information

If the `ttApplicationContext` built-in procedure was used to encode context information in an XLA record, that information is included in the `__CONTEXT` name/value pair in the `MapMessage`. If no context information is provided, the `__CONTEXT` value is not included in the `MapMessage`.

DDL event data formats

Many Data Definition Language (DDL) operations generate XLA updates that can be monitored by XLA event handlers. This section describes the contents of the `MapMessage` objects that are generated for these operations.

CREATE_TABLE

Messages with `__TYPE=1 (XlaConstants.CREATE_TABLE)` indicate that a table has been created. Table 8.3 shows the name/value pairs that are included in a `MapMessage` generated for a CREATE_TABLE operation.

Table 8.3 Create Table Data Provided in Update Messages

Name	Value
`OWNER`	String value of the owner of the created table.
`NAME`	String value of the name of the created table.
`PK_COLUMNS`	String value containing the names of the columns in the primary key for this table. If the table has no primary key, the PK_COLUMNS value is not specified. Format: `<col1name>[;<col2name>[;<col3name>[;…]]]`

Table 8.3 Create Table Data Provided in Update Messages

Name	Value
COLUMNS	String value containing the names of the columns in the table. Format: <col1name>[;<col2name>[;<col3name>[;…]]] **Note:** For each column in the table, additional name/value pairs that describes the column are included in the MapMessage.
*_column_name*_TYPE	Integer value representing the datatype of this column. From java.sql.types.
*_column_name*_PRECISION	Integer value containing the precision of this column (for NUMERIC / DECIMAL).
*_column_name*_SCALE	Integer value containing the scale of this column (for NUMERIC / DECIMAL).
*_column_name*_SIZE	Integer value indicating the maximum size of this column (for CHAR / VARCHAR / BINARY / VARBINARY).
*_column_name*_NULLABLE	Boolean value indicating whether this column can have a NULL value.
*_column_name*_OUTOFLINE	Boolean value indicating whether this column is stored in the “inline” or “out of line” part of the tuple.
*_column_name*_INPRIMARYKEY	Boolean value indicating whether this column is part of the primary key of the table.

DROP_TABLE

Messages with __TYPE=2 (XlaConstants.DROP_TABLE) indicate that a table has been dropped. Table 8.4 shows the name/value pairs that are included in a

MapMessage generated for a DROP_TABLE operation.

Table 8.4 Drop Table Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the sequence.
NAME	String value of the name of the dropped sequence.

CREATE_INDEX

Messages with __TYPE=3 (XlaConstants.CREATE_INDEX) indicate that an index has been created. Table 8.5 shows the name/value pairs that are included in a MapMessage generated for a CREATE_INDEX operation.

Table 8.5 Create Index Data Provided in Update Messages

Name	Value
TBLOWNER	String value of the owner of the table on which the index was created.
TBLNAME	String value of the name of the table on which the index was created.
IXNAME	String value of the name of the created index.
INDEX_TYPE	String value representing the index type: “P” (Primary Key), “F” (Foreign Key), or “R” (Regular).
INDEX_METHOD	String value representing the index method: “H” (Hash) or “T” (T-tree).
UNIQUE	Boolean value indicating whether or not the index is UNIQUE.

Table 8.5 Create Index Data Provided in Update Messages

Name	Value
HASH_PAGES	Integer value representing the number of PAGES in a hash index. (Not specified for T-Tree indexes).
COLUMNS	String value describing the columns in the index. Format: <col1name>[;<col2name>[;<col3name>[;…]]]

DROP_INDEX

Messages with __TYPE=4 (XlaConstants.DROP_INDEX) indicate that an index has been dropped. Table 8.6 shows the name/value pairs that are included in a MapMessage generated for a DROP_INDEX operation.

Table 8.6 Drop Index Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the table on which the index was dropped.
TABLE_NAME	String value of the name of the table on which the index was dropped.
INDEX_NAME	String value of the name of the dropped index.

ADD_COLUMNS

Messages with `__TYPE=5 (XlaConstants.ADD_COLUMNS)` indicate that a table has been altered by adding new columns. Table 8.7 shows the name/value pairs that are included in a `MapMessage` generated for a ADD_COLUMNS operation.

Table 8.7 Add Columns Data Provided in Update Messages

Name	Value
`OWNER`	String value of the owner of the altered table.
`NAME`	String value of the name of the altered table.
`PK_COLUMNS`	String value containing the names of the columns in the primary key for this table. If the table has no primary key, the PK_COLUMNS value is not specified. Format: `<col1name>[;<col2name>[;<col3name>[;…]]]`
`COLUMNS`	String value containing the names of the columns added to the table. Format: `<col1name>[;<col2name>[;<col3name>[;…]]]` **Note:** For each added column, additional name/value pairs that describe the column are included in the `MapMessage`.
`_column_name``_TYPE`	Integer value representing the datatype of this column. From `java.sql.types`.
`_column_name``_PRECISION`	Integer value containing the precision of this column (for NUMERIC / DECIMAL).
`_column_name``_SCALE`	Integer value containing the scale of this column (for NUMERIC / DECIMAL).
`_column_name``_SIZE`	Integer value indicating the maximum size of this column (for CHAR / VARCHAR / BINARY / VARBINARY).

Table 8.7 Add Columns Data Provided in Update Messages

Name	Value
*_column_name*_NULLABLE	Boolean value indicating whether this column can have a NULL value.
*_column_name*_OUTOFLINE	Boolean value indicating whether this column is stored in the “inline” or “out of line” part of the tuple.
*_column_name*_INPRIMARYKEY	Boolean value indicating whether this column is part of the primary key of the table.

DROP_COLUMNS

Messages with __TYPE=6 (XlaConstants.DROP_COLUMNS) indicate that a table has been altered by dropping existing columns. Table 8.8 shows the name/value pairs that are included in a MapMessage generated for a DROP_COLUMNS operation.

Table 8.8 Drop Columns Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the altered table.
NAME	String value of the name of the altered table.
COLUMNS	String value containing the names of the columns dropped from the table. Format: <col1name>[;<col2name>[;<col3name>[;…]]] **Note:** For each dropped column, additional name/value pairs that describe the column are included in the MapMessage.
*_column_name*_TYPE	Integer value representing the datatype of this column. From java.sql.types.

Table 8.8 Drop Columns Data Provided in Update Messages

Name	Value
*_column_name*_PRECISION	Integer value containing the precision of this column (for NUMERIC / DECIMAL).
*_column_name*_SCALE	Integer value containing the scale of this column (for NUMERIC / DECIMAL).
*_column_name*_SIZE	Integer value indicating the maximum size of this column (for CHAR / VARCHAR / BINARY / VARBINARY).
*_column_name*_NULLABLE	Boolean value indicating whether this column can have a NULL value.
*_column_name*_OUTOFLINE	Boolean value indicating whether this column is stored in the “inline” or “out of line” part of the tuple.
*_column_name*_INPRIMARYKEY	Boolean value indicating whether this column is part of the primary key of the table.

CREATE_VIEW

Messages with `__TYPE=14 (XlaConstants.CREATE_VIEW)` indicate that a materialized view has been created. Table 8.9 shows the name/value pairs that are included in a `MapMessage` generated for a CREATE_VIEW operation.

Table 8.9 Create View Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the created view.
NAME	String value of the name of the created view.

DROP_VIEW

Messages with __TYPE=15 (XlaConstants.DROP_VIEW) indicate that a materialized view has been dropped. Table 8.9 shows the name/value pairs that are included in a MapMessage generated for a DROP_VIEW operation

Table 8.10 Drop View Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the dropped view.
NAME	String value of the name of the dropped view.

CREATE_SEQ

Messages with __TYPE=16 (XlaConstants.CREATE_SEQ) indicate that a SEQUENCE has been created. Table 8.11 shows the name/value pairs that are included in a MapMessage generated for a CREATE_SEQ operation

Table 8.11 Create Sequence Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the created sequence.
NAME	String value of the name of the created sequence.
CYCLE	Boolean value indicating whether the CYCLE option was specified on the new sequence.
INCREMENT	Long value indicating the INCREMENT BY option specified for the new sequence.
MIN_VALUE	Long value indicating the MINVALUE option specified for the new sequence.
MAX_VALUE	Long value indicating the MAXVALUE option specified for the new sequence.

DROP_SEQ

Messages with __TYPE=17 (XlaConstants.DROP_SEQ) indicate that a sequence has been dropped. Table 8.12 shows the name/value pairs that are included in a

MapMessage generated for a DROP_SEQ operation

Table 8.12 Drop Sequence Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the dropped table.
NAME	String value of the name of the dropped table.

TRUNCATE

Messages with __TYPE=18(XlaConstants.TRUNCATE) indicate that a table has been truncated. All rows in the table have been deleted. Table 8.13 shows the name/value pairs that are included in a MapMessage generated for a TRUNCATE operation

Table 8.13 Truncate Data Provided in Update Messages

Name	Value
OWNER	String value of the owner of the truncated table.
NAME	String value of the name of the truncated table.

Data type mapping

TimesTen Column Type	JMS Message Type	Read With
CHAR(n)	String	MapMessage.getString
VARCHAR(n)	String	MapMessage.getString
NCHAR(n)	String	MapMessage.getString
NVARCHAR(n)	String	MapMessage.getString
DOUBLE	double	MapMessage.getDouble
FLOAT	float	MapMessage.getFloat
DECIMAL(p,s)	String	MapMessage.getString. Can be converted to BigDecimal or to double by the application.
NUMERIC(p,s)	String	MapMessage.getString. Can be converted to BigDecimal or to double by the application.
INTEGER	int	MapMessage.getInt
SMALLINT	short	MapMessage.getShort
TINYINT	short	MapMessage.getShort
BIGINT	long	MapMessage.getLong
BINARY(n)	byte[]	MapMessage.getBytes
VARBINARY(n)	byte[]	MapMessage.getBytes
DATE	String	MapMessage.getString. Can be converted to Date or Calendar by the application.
TIME	String	MapMessage.getString. Can be converted to Date or Calendar by the application.
TIMESTAMP	String	MapMessage.getString. Can be converted to Date or Calendar by the application.

JMS Classes

You can use the JMS classes shown in Table 8.14 when programming to the JMS/XLA API. The JMS/XLA API only supports publish/subscribe messaging.

Table 8.14 JMS/XLA Classes

Message (parent class only)	**MapMessage**
TopicConnectionFactory	**TopicConnection**

Table 8.14 JMS/XLA Classes

Topic	**TopicSession**
TopicSubscriber	**ConnectionFactory**
Connection	**Destination**
Session	**MessageConsumer**
ConnectionMetaData	**ExceptionListener**

JMS Message Header Fields

Table 8.15 shows the JMS Message Header Fields provided by JMS/XLA.

Table 8.15 JMS/XLA Header Fields

Header	Contents
`JMSMessageId`	The LFN/LFO values from the corresponding XLA record.
`JMSType`	The string representation of the `__TYPE` field

Glossary

.odbc.ini file

See ODBC.INI file.

ACID transaction semantics

An acronym referring to the four fundamental properties of a transaction: atomicity, consistency, isolation and durability.

API

See "application programming interface (API)".

application programming interface (API)

A set of functions that an application uses to request and carry out lower-level services. The ODBC API is composed of the ODBC functions.

atomicity

A property of a transaction whereby either all or none of the operations of a transaction are applied to the data store.

backup instance

A set of files containing backup information for a given data store, residing at a given "backup path". See also "backup point", "full backup" and "incremental backup".

backup path

The location of a data store, specified by a directory name and an optional basename.

backup point

The time at which a backup begins. See also "backup path", "full backup" and "incremental backup".

cache group

A set of cached tables related through foreign keys.

cache instance

A set of rows related through foreign keys. Each cache instance contains exactly one row from the root table of a cache group and zero or more rows from the other tables in the cache group.

CallableStatement

A type of JDBC statement object used for calls to built-in procedures.

class

A Java language prototype that defines the variables and the methods common to all objects of a certain kind.

client/server

An approach to application design and development in which application processing is divided between components running on an end user's machine (the client) and a network server. Generally, user interface elements are implemented in the client component, while the server controls database access.

client data source name

See "data source name, client".

concurrency

The ability to have multiple transactions access and manipulate the data store at the same time.

connection

A data path between an application and a particular ODBC data source.

connection attribute

A character string that defines a connection parameter to be used when connecting to an ODBC data source. Connection attributes have the form *name=value*, where *name* is the name of the parameter and *value* is the parameter value. See also connection string.

connection request

A message sent by an application through an ODBC driver to an ODBC data source to request a connection to that data source.

connection string

A character string that defines the connection parameters to be used when connecting to an ODBC data source. A connection string is expressed as one or more connection attributes separated by semicolons.

consistency

A property of transactions whereby each transaction transforms the database from one consistent state to another.

cursor

A control structure used by an application to iterate through the results of an SQL query.

data source definition

A named collection of connection attributes that defines the connection parameters to be used when connecting to an ODBC data source. See also "data source name".

data source name

A logical name by which an end user or application refers to an ODBC data source definition. Sometimes incorrectly used to mean "data source definition." See also "data source definition", ODBC.INI file.

data source name, client

A data source name defined on a TimesTen client machine that refers to a TimesTen server data source name on a server machine.

data source name, server

A system data source name (system DSN) defined on a server machine. Server Data Source Names become available to all TimesTen clients on a network when the TimesTen Server is running.

data source name, system

A data source name that is accessible by all users of a particular machine.

data source name, user

A data source name that is accessible only by the user who created the data source name.

driver

See "ODBC driver".

DSN

See "data source name".

DSN, client

See "data source name, client".

DSN, server

See "data source name, server".

DSN, system

See "data source name, system".

DSN, user

See "data source name, user".

durability

A property of transactions whereby the effects of a committed transaction survive system failures.

environment variable

A *name*, *value* pair maintained by the operating system that can be used to pass configuration parameters to an application.

event

An activity or occurrence that can be tracked by a logging mechanism in an application, service or operating system. See also "logging", "protocol message logging" and "event viewer".

event viewer

On Windows, a utility program used to view the contents of the operating system event log.

firewall

A combination of hardware and software that reliably separates one part of a network from another for security purposes.

full backup

A data store backup procedure in which a complete copy of a data store is created. Typically, the first backup of a data store must be a full backup. See also "incremental backup".

host

A computer. Typically used to refer to a computer on a network that provides services to other computers on the network.

host name

A character string name that uniquely identifies a particular computer on a network. Examples: `athena`, `thames.mycompany.com`. See also "host".

in-line column

A column whose values are physically stored together with the other column values of a row.

incremental backup

A data store backup procedure in which an existing backup is augmented with all the log records created since its last full or incremental backup. See also "backup instance" and "full backup".

initialization file

See ODBC.INI file.

IP address

A numeric address that uniquely identifies a computer on a network and consists of four numbers separated by dots. Abbreviation for Internet Protocol address. Example: `123.61.129.91`.

IPC

Inter Process Communication.

isolation

A property of transactions whereby each transaction runs as if it were the only transaction in the system.

listener thread

A thread that runs on the TimesTen Server that receives and processes connection requests from TimesTen Clients.

Java Database Connectivity (JDBC)

A standard Java language SQL database API that provides uniform access to a wide range of relational databases. It also provides a common base on which higher level tools and interfaces can be built.

JDBC

See "Java Database Connectivity (JDBC)".

logging

The process by which an application, service or operating system records specific events that occur during processing.

method

A Java language procedure or routine defined in a class.

multi-threading

A programming paradigm in which a process contains multiple threads of control.

network address

A host name, or IP address that uniquely identifies a particular computer on a network. Examples: `123.61.129.91`, `athena`, `thams.mycompany.com`.

ODBC

See "Open Database Connectivity (ODBC)".

ODBC Administrator

A utility program used on Windows to create, configure and delete data source definitions.

ODBC data source

See "data source name" (DSN).

ODBC data source name

See "data source name" (DSN).

ODBC driver

A library that implements the function calls defined in the ODBC API and enables applications to interact with ODBC data sources.

ODBC Driver Manager

A library that acts as an intermediary between an ODBC application and one or more ODBC drivers.

ODBC initialization file

See ".odbc.ini file".

Open Database Connectivity (ODBC)

A database-independent application programming interface that enables applications to access data stored in heterogeneous relational and non-relational databases. Based on the Call-Level Interface (CLI) specification developed by X/Open's SQL Access Group and first popularized by Microsoft on the Windows platform.

out-of-line column

A column whose values are physically stored separately from the other column values of a row.

phantom

A row that appears during one read but not during another read within the same transaction, due to the actions of other concurrently executing transactions.

ping

A utility that tests the connection between two computers on a network by sending a message from one computer to the other and measuring how long it takes for the receiving system to confirm that the message was received. Typically packaged with network software.

port number

See "TCP/IP port number".

procedure

See "stored procedure".

process

An instance of a program in execution.

propagate

When using Cache Connect to Oracle, to send table or row modifications from a Cache Connect to Oracle data store to an Oracle database. Compare with "replicate".

protocol message logging

The process that the TimesTen Server uses to record each message it receives through the TimesTen network protocol.

relational database

A type of database or DBMS that stores data in tables organized as rows and columns. A relational database can perform searches by using data in specified columns of one table to find data stored in another table.

replicate

The sending of table or row modifications from one data store to another. Compare with "propagate".

result set

A collection of zero or more rows of data that represent the result of an SQL query.

rollback

To undo the actions of a transaction, thereby returning all items modified by the transaction to their original state.

row buffering

A performance enhancement used by the TimesTen Client in which the client receives multiple result rows of an SQL query in each message from the TimesTen Server to reduce network communication.

RPC

Remote Procedure Call.

scalability

The degree to which a system or application can handle increasing demands on system resources without significant performance degradation.

server data source name

See "data source name, server".

server DSN

See "data source name, server".

system DSN

See "data source name, system".

shorthand name

A logical name used to refer to a particular TimesTen Server. Shorthand names relieve the end user of having to enter a host name and port number to connect to a TimesTen Server.

SMP

Symmetric multi-processing. A hardware configuration in which two or more similar processors are connected via a high-bandwidth link and managed by one operating system, where each processor has equal access to I/O devices.

SNMP

Simple Network Management Protocol. Used to manage nodes on a network.

SQL

Structured Query Language.

stack overflow condition

An error condition in which the stack usage of a thread or process exceeds the amount of space allocated for the stack.

stored procedure

An executable object or named entity stored in a data store that can be invoked with input and output parameters and which can return result sets similar to those returned by an SQL query.

system account

A special account on Windows used by the operating system and certain operating system services. The TimesTen Service and the TimesTen Server run under the system account.

system DSN

See "data source name, system".

T-tree

An index structure similar in functionality to a B-tree but optimized for in-memory data management.

TCP/IP

The communications protocol used by computers on the Internet. Abbreviation for Transport Control Protocol/Internet Protocol.

TCP/IP port number

A number used by TCP/IP that identifies the end point for a connection to a host that supports multiple simultaneous connections.

telnet

A utility program and protocol that enables a user on one computer to open a virtual terminal, log in to a remote host and interact as a terminal user of that host.

thread

An independent sequence of execution of program code inside a process. See also "process".

thread-safe ODBC driver

An ODBC driver that supports multithreaded servers and clients. The TimesTen data manager driver and the TimesTen Client driver are thread-safe.

timeout error

An error condition indicating that the requested operation did not complete within the given amount of time. See also "timeout interval".

timeout interval

A configuration parameter that specifies the maximum amount of time that an operation should take to complete. See also "timeout error".

TimesTen Client

(1) An ODBC driver that enables end users to access data sources through a TimesTen Server. (2) A computer on which the TimesTen Client software has been installed. Using the TimesTen Client driver, an end user or application can access any data source managed by an available TimesTen Server.

TimesTen Client/Server network protocol

The protocol used by TimesTen Clients and TimesTen Servers to exchange data over a standard TCP/IP network connection.

TimesTen Server

(1) An application program that makes TimesTen data sources available to the TimesTen Clients on a network. (2) A computer on which the TimesTen Server software is running.

TimesTen Server address

The host name or IP address used during installation of the TimesTen Server to identify the computer on which the software is being installed.

transaction

An operation or set of operations performed against data in a data store. The operation(s) defined in a transaction must be completed as a whole; if any part of the transaction fails, the entire transaction fails. See also "ACID transaction semantics".

UTF-8

A variable-length encoding scheme in which each Unicode character is represented by a sequence of bytes ranging in length from 1 to 4 bytes. Among other properties, this encoding has the property that the Unicode ASCII range values are encoded as single-byte ASCII characters.

UTF-16

An encoding scheme defined by the ISO/IEC 10646 standard in which each Unicode character is represented by either a two-byte integer or a pair of two-

byte integers. Characters from European scripts and most Asian scripts are represented in two bytes. Surrogate pairs are represented in four bytes. Surrogate pairs represent characters such as infrequently used Asian characters that were not included in the original range of two-byte characters.

user account

The combination of a user name, password and access permissions that gives an individual user access to an operating system.

user data source name

See "data source name, user".

user DSN

See "data source name, user".

User Manager

A Windows utility program used to create user accounts and assign access rights and group membership.

Windows sockets (Winsock)

An API that defines a standard binary interface for TCP/IP transports on Windows platforms. This API adds Windows-specific extensions to the Berkeley Sockets interface originally defined in Berkeley UNIX.

Index

T

U

W

X