Hewlett Packard
Enterprise

HPE NonStop JDBC Type 4 Driver Programmer's Reference for SQL/MX Release 3.2.1

Part Number: 691128-007R
Published: November 2015
Edition: J06.15 and subsequent J-series RVUs and H06.26 and subsequent H-series RVUs

The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Links to third-party websites take you outside the Hewlett Packard Enterprise website. Hewlett Packard Enterprise has no control over and is not responsible for information outside the Hewlett Packard Enterprise website.

Acknowledgments

Microsoft®, Windows® and Windows NT® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Intel®, Itanium®, Pentium®, and Celeron® are trademarks or registered trademarks of Intel Corporation in the United States and other countries.

UNIX® is a registered trademark of The Open Group.

Motif, OSF/1, X/Open, and the "X" device are registered trademarks and IT DialTone and The Open Group are trademarks of The Open Group in the U.S. and other countries.

Open Software Foundation, OSF, the OSF logo, OSF/1, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc.

OSF MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE OSF MATERIAL PROVIDED HEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

OSF shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

© 1990, 1991, 1992, 1993 Open Software Foundation, Inc. This documentation and the software to which it relates are derived in part from materials supplied by the following:

© 1987, 1988, 1989 Carnegie-Mellon University. © 1989, 1990, 1991 Digital Equipment Corporation. © 1985, 1988, 1989, 1990 Encore Computer Corporation. © 1988 Free Software Foundation, Inc. © 1987, 1988, 1989, 1990, 1991 Hewlett-Packard Company. © 1985, 1987, 1988, 1989, 1990, 1991, 1992 International Business Machines Corporation. © 1988, 1989 Massachusetts Institute of Technology. © 1988, 1989, 1990 Mentat Inc. © 1988 Microsoft Corporation. © 1987, 1988, 1989, 1990, 1991, 1992 SecureWare, Inc. © 1990, 1991 Siemens Nixdorf Informationssysteme AG. © 1986, 1989, 1996, 1997 Sun Microsystems, Inc. © 1989, 1990, 1991 Transarc Corporation.

This software and documentation are based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. OSF acknowledges the following individuals and institutions for their role in its development: Kenneth C.R.C. Arnold, Gregory S. Couch, Conrad C. Huang, Ed James, Symmetric Computer Systems, Robert Elz. © 1980, 1981, 1982, 1983, 1985, 1986, 1987, 1988, 1989 Regents of the University of California.

Java® and Oracle,® are registered trademarks of Oracle and/or its affiliates.

Printed in the U.S.

Contents

About this document

This document describes how to use the HPE NonStop JDBC Type 4 Driver for SQL/MX Release 3.2.1. This driver provides Java applications running on a foreign platform with JDBC access to an HPE NonStop SQL/MX database running on the HPE NonStop platform. Where applicable, the Type 4 driver conforms to the standard JDBC 3.0 API from Oracle.

Product Version

NonStop JDBC Type 4 Driver for SQL/MX Release 3.2.1

Supported Release Version Updates (RVUs)

This publication supports J06.15 and all subsequent J-series RVUs and H06.26 and all subsequent H-series RVUs, until otherwise indicated by its replacement publications. Additionally, all considerations for H-series throughout this manual will hold true for J-series also, unless mentioned otherwise.

Audience

HPE NonStop JDBC Type 4 Driver Programmer's Reference for SQL/MX Release 3.2.1 is for experienced Java programmers who want to access NonStop SQL/MX with the Type 4 driver.

You must be familiar with the following topics:

- J2EE or Java applications running on JDK 1.5 or later
- The JDBC 3.0 specification as defined by Oracle
- JDBC 3.0 API

New and Changed Information

New and Change Information in the November 2015

The following is an update in this editing (691128-007R)

Updated Hewlett Packard Enterprise references.

New and Changed Information in the March 2015

The following is the update in this edition (691128-007):

Updated `fetchBufferSize` range in “fetchBufferSize” (page 44) section.

New and Changed Information in the December 2014

The following is the update in this edition (691128-006):

Added a note at the end of “JDBC Data Types” (page 81) section.

New and Changed Information in the November 2013

The following is the update in this edition (691128-005):

Added a new “autoCommit” (page 55) section.

New and Changed Information in the August 2013

The following are the updates in this edition (691128-003):

- Added a new section for SSL support, “sslEncryption” (page 53).

New and Changed Information in the February 2013

The following are the updates in this edition (691128-002):

- Added a new property “fetchBufferSize” (page 44).
- Updated “Enabling MFC” (page 73).

New and Changed Information in the August 2012

The following are the updates in this edition (691128-001):

- Added “mfuStatementCache” (page 47))
  Updated “Module File Caching (MFC)” (page 73)
- Added “Providing a secure JDBC connection using NonStop SSL” (page 112)

Document Organization

Table 1 (page 9) summarizes the contents of this guide.

Table 1 Summary of Contents

Section	Description
“Introduction to NonStop JDBC Type 4 Driver” (page 15)	Describes the driver architecture.
“Installing and Verifying the Type 4 Driver” (page 18)	Describes where to find information about the installation requirements and explains how to verify the Type 4 driver installation.
“Accessing SQL Databases with SQL/MX” (page 21)	Explains how to access NonStop SQL/MX databases by using the Type 4 driver.
“Type 4 Driver Properties” (page 36)	Describes how to set the properties that configure the driver and provides a detailed description of each property.
“Working with BLOB and CLOB Data” (page 57)	Describes working with BLOB and CLOB data in JDBC applications using the standard interface described in the JDBC 3.0 API specification.
“Managing the SQL/MX Tables for BLOB and CLOB Data” (page 67)	Describes the database management (administrative) tasks for adding and managing the tables for BLOB and CLOB data. The Type 4 driver uses SQL/MX tables in implementing support for BLOB and CLOB data access.
“Module File Caching (MFC)” (page 73)	Explains the Module File Caching (MFC) feature.
“Type 4 Driver Compliance” (page 76)	Explains how the Type 4 driver differs from the Oracle JDBC 3.0 standard. Lists supported features of SQL/MX 3.2, unsupported features for NonStop SQL/MX and NonStop SQL/MP.
“Tracing and Logging Facilities” (page 85)	Explains the tracing and logging facilities.
“Migration and Compatibility” (page 89)	Describes application changes needed for migration to the Type 4 driver.
“Messages” (page 90)	Lists messages from the Type 4 driver. The descriptions include the message-text, the cause, the effect, and recovery information.

Table 1 Summary of Contents *(continued)*

Section	Description
“Providing a secure JDBC connection using NonStop SSL” (page 112)	Describes the procedures to install NonStop SSL.
“Sample Programs Accessing CLOB and BLOB Data” (page 117)	Lists sample programs that show accessing CLOB and BLOB data.

Related Documentation

“Type 4 Driver APIs” (page 10)

“NonStop System Computing Documents” (page 11)

In addition to the standard documentation, white papers about using the Type 4 driver with various application servers and how to improve performance are available in both the product installation and the NonStop Technical Library at Hewlett Packard Enterprise Support Center (HPESC).

Type 4 Driver APIs

Oracle Documents

The following documents are available on Oracle websites:

NOTE: Oracle Java 2 SDK, Standard Edition Documentation Version 1.5 is provided on the NonStop Server for Java 4 product distribution CD in an executable file for your convenience in case you cannot get Java documentation from the Oracle websites. The links to Oracle Java documentation in the JDBC Driver for NonStop SQL/MX documentation go to the Oracle websites, which provide more extensive documentation than SDK 1.5. Hewlett Packard Enterprise cannot guarantee the availability of the J2SE SDK 1.5 documentation on the Oracle websites. Also, Hewlett Packard Enterprise is not responsible for the links or content in the documentation from Oracle.

- JDBC 3.0 Specification, available for downloading from Oracle.
- JDBC API Documentation , includes links to APIs and Tutorials.
- JDBC Data Access API, general information (http://www.oracle.com/technetwork/java/overview-141217.html#1)
- JDBC Data Access API, FAQs for JDBC 3.2
- JDBC API Comments
  - Core JDBCore JDBC 3.2 API in the java.sql package
  - Optional JDBC 3.2 API in the javax.sql package

Type 4 Driver 3.1 Implementation-Specific APIs

For information about Type 4 driver implementation-specific APIs, see the following Type 4 Driver document that is available in the NonStop Technical Library at the Hewlett Packard Enterprise Support Center.

NonStop System Computing Documents

The following are the manuals in the SQL/MX customer library:

- **Introductory guidesTHan**

SQL/MX Comparison Guide for SQL/MP Users	Describes differences between NonStop SQL/MP and NonStop SQL/MX databases.
SQL/MX Quick Start Guide	Describes basic techniques for using SQL in the SQL/MX conversational interface (MXCI). Includes information about installing the sample database.

- **Installation guides**

SQL/MX Installation and Upgrade Guide	Describes how to plan for, install, create, and upgrade a SQL/MX database.
SQL/MX Management Manual	Describes how to manage a SQL/MX database.
NSM/web Installation Guide	Describes how to install NSM/web and troubleshoot NSM/web installations.

- **Reference manuals**

SQL/MX Reference Manual	Describes the syntax of SQL/MX statements, MXCI commands, functions, and other SQL/MX language elements.
SQL/MX Messages Manual	Describes SQL/MX messages.
SQL/MX Glossary	Defines SQL/MX terminology.

- **Connectivity manuals**

SQL/MX Connectivity Service Manual	Describes how to install and manage SQL/MX Connectivity Service (MXCS), which enables ODBC and other connectivity APIs to use NonStop SQL/MX.
SQL/MX Connectivity Service Administrative Command Reference	Describes the SQL/MX Administrative Command Library (MACL) available with the SQL/MX conversational interface (MXCI).
ODBC/MX Driver for Windows	Describes how to install and configure HPE NonStop ODBC/MX for Microsoft Windows, which enables applications developed with ODBC API to use NonStop SQL/MX.
SQL/MX Remote Conversational Interface (RMXCI) Guide	Describes how to use SQL/MX Remote Conversational Interface to run the RMXCI commands, and SQL statements interactively or from script files.
HPE NonStop MXDM User Guide	Describes how to use the NonStop SQL/MX Database Manager (MXDM) to monitor and manage the SQL/MX database.
HPE NonStop JDBC Type 2 Driver Programmer's Reference	Describes the NonStop JDBC Type 2 Driver functionality, which allows Java programmers to remotely develop applications deployed on client workstations to access NonStop SQL/MX databases.
HPE NonStop MXDM User Guide for SQL/MX Release 3.2.1	Describes how to use the NonStop SQL/MX Database Manager (MXDM) to monitor and manage the SQL/MX database.

- **Migration guides**

SQL/MX Database and Application Migration Guide	Describes how to migrate databases and applications to NonStop SQL/MX, and how to manage different versions of NonStop SQL/MX.
NonStop NS-Series Database Migration Guide	Describes how to migrate NonStop SQL/MX, NonStop SQL/MP, Enscribe databases and applications to HPE Integrity NonStop NS-series systems.

- **Data management guides**

SQL/MX Data Mining Guide	Describes the SQL/MX data structures and operations for data mining.
SQL/MX Report Writer Guide	Describes how to produce formatted reports using data from an SQL/MX database.
DataLoader/MX Reference Manual	Describes the features and functions of the DataLoader/MX product, a tool to load SQL/MX databases.

- **Application development guides**

SQL/MX Programming Manual for C and COBOL	Describes how to embed SQL/MX statements in ANSI C and COBOL programs.
SQL/MX Query Guide	Describes how to understand query execution plans and write optimal queries for an SQL/MX database.
SQL/MX Queuing and Publish/Subscribe Services	Describes how NonStop SQL/MX integrates transactional queuing and publish/subscribe services into its database infrastructure.
SQL/MX Guide to Stored Procedures in Java	Describes how to use stored procedures that are written in Java within NonStop SQL/MX.

- **Online help**

SQL/MX Database Manager Help	Contents and reference entries from the SQL/MX Database Manager User Guide.
Reference Help	Overview and reference entries from the SQL/MX Reference Manual.
Messages Help	Individual messages grouped by source from the SQL/MX Messages Manual.
Glossary Help	Terms and definitions from the SQL/MX Glossary.
NSM/web Help	Context-sensitive help topics that describe how to use the NSM/web management tool.
Visual Query Planner Help	Context-sensitive help topics that describe how to use the Visual Query Planner graphical user interface.

The NSM/web, SQL/MX Database Manager, and Visual Query Planner help systems are accessible from their respective applications. You can download the Reference, Messages, and Glossary online help from the Hewlett Packard Enterprise Software Depot, at http://www.hpe.com/support/softwaredepot. For more information about downloading online help, see the *SQL/MX Release 3.2.1 Installation and Upgrade Guide*.

Notation Conventions

Hypertext Links

Blue underline is used to indicate a hypertext link within text. By clicking a passage of text with a blue underline, you are taken to the location described. For example:

This requirement is described under Backup DAM Volumes and Physical Disk Drives.

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS. Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters. Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

computer type. `Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type. `Italic computer type` letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

```
pathname
```

[ ] Brackets. Brackets enclose optional syntax items. For example:

```
TERM [\system-name.]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num ]
```

```
[ -num ]
```

```
[ text ]
```

```
K [ X | D ] address
```

{ } Braces. A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
```

```
{ $process-name }
```

```
ALLOWSU { ON | OFF }
```

| Vertical Line. A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis. An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address [ , new-value ]…
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}…
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char…"
```

Punctuation. Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing. Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing. If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE
[ , attribute-spec ]…
```

Publishing History

Part Number	Product Version	Published
640331-001	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.0	February 2011
663860-001	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.1	October 2011
691128-001	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2	August 2012
691128-002	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2.1	February 2013
691128-003	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2.1	August 2013
691128–005	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2.1	November 2013
691128–006	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2.1	December 2014
691128–007	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2.1	March 2015
691128–007R	JDBC/MX Type 4 Driver for NonStop SQL/MX Release 3.2.1	November 2015

1 Introduction to NonStop JDBC Type 4 Driver

Type 4 Driver API Package

The Type 4 driver package, `com.tandem.t4jdbc`, is shipped with the driver software. For class and method descriptions, see the *HPE NonStop JDBC Type 4 Driver API Reference manual* in the NonStop Technical Library at http://www.hpe.com/info/nonstop-docs.

The NonStop JDBC Type 4 Driver (here after Type 4 driver) implements JDBC technology that conforms to the standard JDBC 3.0 Data Access API. This driver enables Java applications to use HPE NonStop SQL/MX Release 3.0 or later versions to access NonStop SQL databases.

To obtain detailed information on the standard JDBC API, you should download the JDBC API documentation provided by Oracle (http://www.oracle.com/technetwork/java/download-141179.html).

Type 4 Driver Architecture

The Type 4 driver uses JDBC 3.0 standard functionality to allow Java clients access to NonStop SQL/MX from client platforms.

The Type 4 driver requires these products on the NonStop server: the HPE NonStop SQL Connectivity Service (MXCS) and NonStop SQL/MX Release 3.0 or later versions, which both require the HPE NonStop Open System Services (OSS) environment.

The SQL/MX Connectivity Service (MXCS) must be running on the NonStop system. For more information, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1.*

For XA transactions, the Type 4 driver requires the HPE NonStop XA Broker product installed on the NonStop server.

Client Platforms Supported

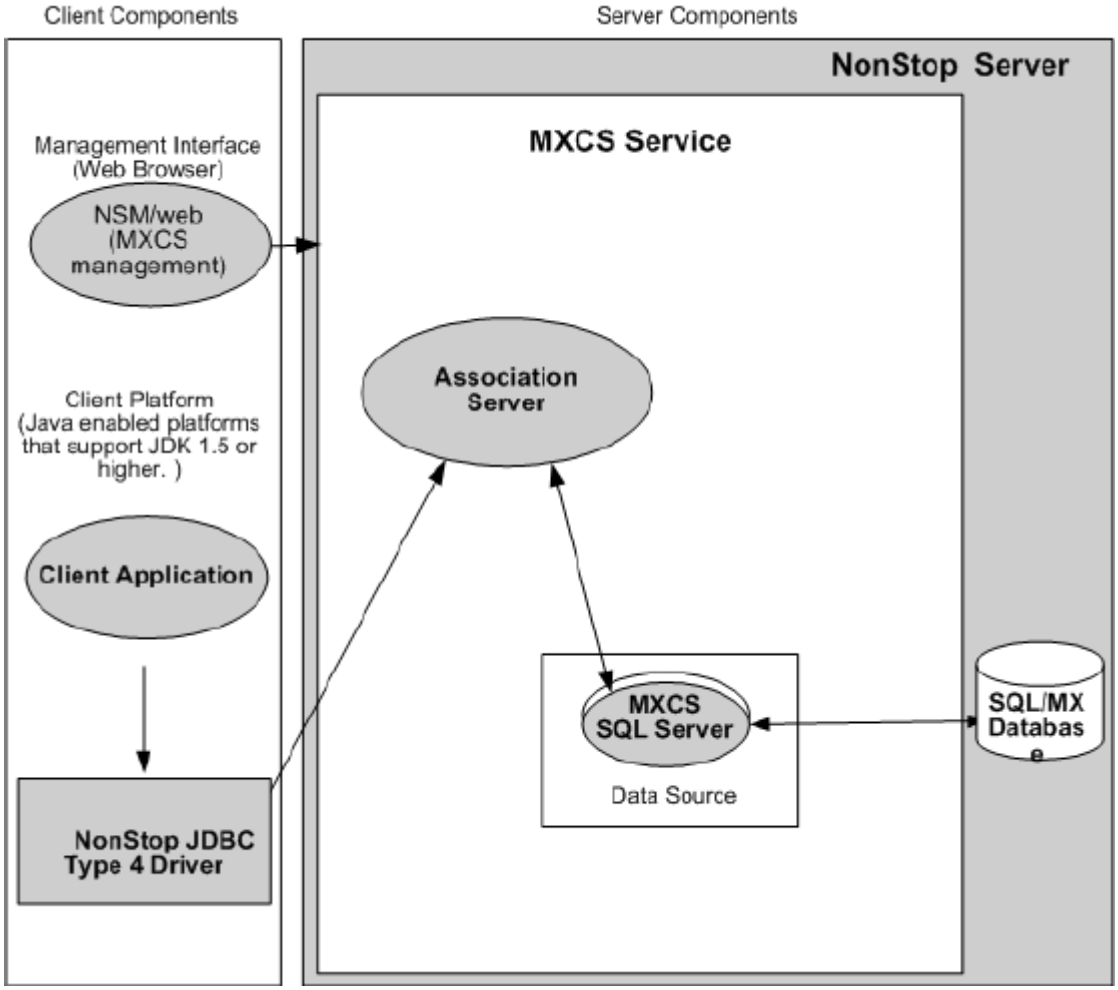
Java enabled platforms that support JDK 1.5 or later. For more information, see the softdoc file delivered with the product.

Accessing a Database with the Type 4 Driver

1. A Java client application establishes a connection.
2. The Type 4 driver accesses an MXCS association server on the NonStop system.
3. The association server returns a handle for an MXCS server process to the Type 4 driver.
4. The driver makes a connection with the MXCS server process and returns that connection to the client.
5. The MXCS process receives JDBC Type 4 driver requests from the client and processes them using the SQL/MX database engine.

The following figure illustrates database access using the Type 4 driver:

Figure 1 Type 4 driver, through the MXCS Service, Accesses an SQL/MX Database



Samples

The following samples are provided with the Type 4 driver.

Sample	Description
CallableStatementSample	Demonstrates the invocation of a stored procedure in Java (SPJ).(1)
DBMetaSample	Gets metadata from a table by using the Connection interface and ResultSetMetaDatainterface. The information retrieved includes Type, (of what getTypeInfo()), catalogs, tables, procedures, columns, columns, column count on tables, and so forth.(1)
PreparedStatementSample	Using the PreparedStatement interface, performs a select all and parameterized select for data type of CHAR, INT, TIMESTAMP, DECIMAL, NUMERIC, and DATE.(1)
ResultSetSample	Executes a simple select statement, metadata information about the result set returned by the simple select statement, and finally, uses the result-set metadata to print out each row returned by the simple select statement.(1)
SQLMPSample	Executes a simple select statement on an SQL/MP table, retrieves metadata information about the result set returned by the simple select statement, and, uses the result-set metadata to print out each row returned by the simple select statement. (Access an existing SQL/MP table through the SQL/MX engine.)

Sample	Description
StatementSample	Executes a simple select statement, retrieves metadata information about the result set returned by the simple select statement, and uses the result-set metadata to print out each row returned by the simple select statement.(1)
WLS_MedRecSample	Demonstrates that a J2EE application built to work with a database like Oracle can work with NonStop SQL/MX by just reconfiguring the application in BEA WebLogic Server. No coding changes or recompilation are required to switch to NonStop SQL/MX from a different relational database management system (RDBMS).(2)
WLS_TwoPhaseandPoolingSample	Demonstrates that a connected pooling, particularly the number physical connections created does not exceed the maximum number of physical connections configured in BEA WebLogicServer 8.1.(2)
WLS_TwoPhaseSample	Demonstrates that a Type 4 driver can be configured as a XA driver exhibiting two-phase commit with BEA WebLogic Server 8.1. The Type 4 driver can participate in a two-phase commit

NOTE:

1. Creates, populates, and drops sample SQL/MX tables.
2. The sample's documentation explains how to create the SQL/MX tables.

For information on these samples, see the README file provided with the Type 4 driver software under the samples directory.

2 Installing and Verifying the Type 4 Driver

Prerequisites

Hardware and software requirements for the JDBC Type 4 driver are described in the Softdoc file delivered with the product. Read that document before installing the product.

You must install and configure the following before installing the JDBC Type 4 driver:

- JDK 1.5 or later in the client environment
- NonStop SQL/MX Release 3.2 on the NonStop system
- The ODBC/MX Service on the NonStop system. For information on starting the MXCS Association Server, see the *HPE NonStop Connectivity Services Manual for SQL/MX Release 3.2.1*.

Installing the JDBC/MX Type 4 Driver

To install the JDBC Type 4 driver:

1. Log on to the NonStop system as `super.super`.
2. Change to the TACL prompt.
3. The JDBC T4 driver is available at `$SYSTEM.ZMXODBC`. To change to the `$SYSTEM.ZMXODBC` subvolume, enter the following:

   ```
   $SYSTEM STARTUP 1> volume $SYSTEM.ZMXODBC
   ```
4. To locate the T1249TAR file:

   ```
   $SYSTEM.ZMXODBC> fileinfo T1249TAR
   ```
5. Using the BINARY mode, transfer the `T1249TAR` file to the client workstation.
6. On the client system, change to the directory that contains the JDBC Type 4 driver tar file.
7. Rename `T1249TAR` to `T1249.tar`.
8. To untar the file:

   ```
   tar -xvf T1249.tar
   ```

   Alternatively, you can use a tool to unzip the tar file.

   The following folders are created:

   - `samples`
   - `docs`
   - `lib`
   - `install`

9. To set up the client environment, configure one of the following paths:

   On the Windows system:

   - `set JAVA_HOME=<`***`JDK directory`***`>`
   - `set PATH=`**`%PATH%;%JAVA_HOME%\bin`**
   - `set CLASSPATH=`**`%CLASSPATH%;<`*`Type 4 driver HOME`*`>\lib\t4sqlmx.jar;`**

   On HP-UX, Solaris, NonStop, or Linux system:

   - `export JAVA_HOME=`**`<`*`JDK directory`*`>`**
   - `export PATH=`**`$PATH:$JAVA_HOME/bin`**
   - `export CLASSPATH=`**`$CLASSPATH:<`*`Type 4 driver HOME`*`>/lib/t4sqlmx.jar:`**

Product Files

The product contains the following files and directories.

NOTE: This is a partial list.

- a tar file `T1249`.tar that contains these product files:
  - `lib`
    - `t4sqlmx.jar`
  - `samples`
    - CallableStatementSample
    - common
    - DBMetaSample
    - PreparedStatementSample
    - README for samples
    - ResultSetSample
    - t4sqlmx.properties
    - SQLMPSample
    - WLS_MedRecSample
    - WLS_TwoPhaseAndPoolingSample
    - WLS_TwoPhaseSample
  - `docs`
    - white papers
  - `install`
    - `product.contents` (test input file for install, uninstall, and platform fixing)

Setting CLASSPATH

Before running JDBC applications, ensure the `CLASSPATH` environment variable includes the `t4sqlmx.jar` file and installation directory. The `t4sqlmx.jar` file is located in the `lib` directory where you installed the product.

Verifying the Type 4 Driver

To verify the version of the Type 4 driver, use the command:

```
java -jar t4sqlmx.jar
```

An output message similar to the following appears:

```
Version procedure: T1249_V32_30AUG12_HP_NONSTOP(TM)_JDBCT4_2012_06_27
```

3 Accessing SQL Databases with SQL/MX

Communication Overview

The Type 4 driver is a three-layer program.

Figure 2 Type 4 driver—the API layer, IDL layer, and the Transport layer— to the MXCS association server

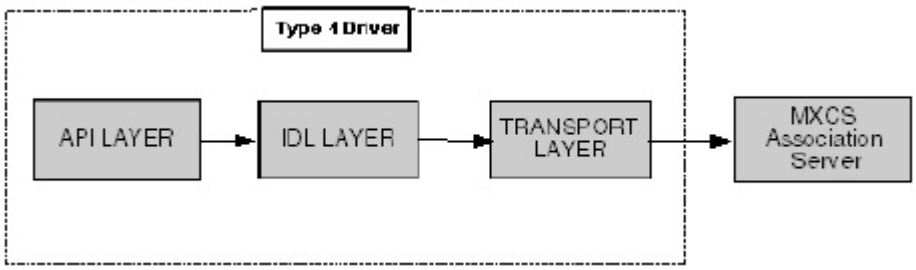


Figure 2 (page 22) illustrates the three layers of the Type 4 driver—the API layer, IDL layer, and the Transport layer— to the MXCS association server.

Data Sources

- "JDBC Data Source (client-side)" (page 22)
- "MXCS Data Source (server-side)" (page 22)

The term data source logically refers to a database or other data storage entity. In this manual, two concepts of data source concepts apply:

- A JDBC (client) data source, which is physically a Java object that contains properties such as the URL of the MXCS Association Server, and the catalog and the schema to use to access the database. The JDBC data source also contains methods for obtaining a JDBC connection to the underlying database.
- An MXCS (server) data source, which is physically a set of information that is created and managed by MXCS. This data source contains configuration information, which defines the semantics of MXCS servers created with that server data source.

JDBC Data Source (client-side)

All JDBC data source classes implement either the `javax.sql.DataSource` interface or the `javax.sql.ConnectionPoolDataSource` interface. The Type 4 driver data source classes are `com.tandem.t4jdbc.SQLMXDataSource` and `com.tandem.t4jdbc.SQLMXDataSource`. (These classes are defined by the JDBC 3.0 specification.) The Type 4 driver implements `javax.sql.XADataSource, java.transaction.xa.XAResource` to support distributed transactions.

Typically, a user or system administrator use a tool to create a data source, and then register the data source by using a JNDI service provider. At run time, a user application typically retrieves the data source through JNDI, and uses the data source's methods to establish a connection to the underlying database.

A `DataSource` object maps to an instance of a database. In the Type 4 driver product, the `DataSource` object acts as an interface between the application code and the database, and enables connection with an MXCS data source.

MXCS Data Source (server-side)

Server data sources reside on a NonStop server. Each server data source represents a pool of SQL MXCS servers that share the same NonStop SQL context. A server data source is typically created by a NonStop system administrator, who defines the semantics of all connections made through that server data source. For example, the server data source contains information on how many MXCS servers can be in its corresponding pool, which defines how many connections can be handled through this data source.

For information on configuring MXCS XA data source, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

Security

Clients connect to the MXCS server with a valid Guardian user ID or alias and password, using standard JDBC 3.0 APIs. An application can make multiple connections to MXCS using different user IDs, and creating different connection objects.

The Type 4 driver provides for user name and password encryption before it is sent to MXCS. The password is encrypted with a proprietary algorithm provided by the MXCS product. For information on secure JDBC connection using NonStop SSL, see “Providing a secure JDBC connection using NonStop SSL” (page 112)

Connecting to SQL/MX

A Java application can obtain a JDBC connection to NonStop SQL/MX in two ways:

- By using the DataSource interface (the preferred method)
- By using the DriverManager class

Connecting with the DataSource Interface

“Overview of Deploying DataSource Objects” (page 23)

“DataSource Object Properties” (page 24)

“Programmatically Creating an Instance of the DataSource Class” (page 24)

“Programmatically Registering the DataSource Object” (page 25)

“Retrieving a DataSource Instance by using JNDI and to the Data Source” (page 25)

“Specifying the Properties File that Configures the Data Source” (page 25)

The `javax.sql.DataSource` interface is the preferred way to establish a connection to the database because this interface enhances the application portability. Portability is achieved by allowing the application to use a logical name for a data source instead of providing driver-specific information in the application. A logical name is mapped to a `javax.sql.DataSource` object through a naming service that uses the Java Naming and Directory Interface (JNDI). Using this `DataSource` method is particularly recommended for application servers.

Observe that two types of data sources interact here as described under Data Sources.

When an application requests a connection by using the `getConnection` method in the `DataSource`, the method returns a Connection object. A `DataSource` object is a factory for `Connection` objects. An object that implements the `DataSource` interface is typically registered with a JNDI service provider.

Overview of Deploying DataSource Objects

Before an application can connect to a `DataSource` object, typically the system administrator deploys the `DataSource` object so that the application programmers can start using it.

Data source properties are usually set by a system administrator using a GUI tool as part of the installation of the data source. Users to the data source do not get or set properties. Management tools can get at properties by using introspection.

Tasks involved in creating and registering a database object are:

1. Creating an instance of the `DataSource` class.
2. Setting the properties of the `DataSource` object.
3. Registering the `DataSource` object with a naming service that uses the Java Naming and Directory Interface (JNDI) API.

An instance of the DataSource class and the DataSource object properties are usually set by an application developer or system administer using a GUI tool as part of the installation of the data source. If you are using an installed data source, skip to topic Programmatically Creating an Instance of a DataSource Object.

The subsequent topics show an example of performing these tasks programmatically.

For more information about using data sources, see the JDBC Tutorial: Chapter 3- Advanced Tutorial http://java.sun.com/developer/Books/JDBCTutorial/index.html or other information available in the field.

DataSource Object Properties

A DataSource object has properties that identify and describe the actual data source that the object represents. These properties include such information as the URL for the MXCS association server, the database schema and catalog names, the location of the database server, the name of the database, and so forth.

NOTE: When a JDBC application uses any server (MXCS) data source, including the default data source, the data source must have been started by MXCS. For more information, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

For details about Type 4 driver properties that you can use with the DataSource object, see “Type 4 Driver Properties” (page 36).

Programmatically Creating an Instance of the DataSource Class

A JDBC application can set DataSource properties programmatically and register with a DataSource object.

To get or set DataSource object properties programmatically, use the appropriate getter or setter methods on the SQLMXDataSource object or the SQLMXConnectionPoolDataSource object. For example,

```
SQLMXDataSource temp = new SQLMXDataSource();
   temp.setCatalog("abc");
```

In the following example, the code fragment illustrates the methods that a DataSource object ds needs to include if the object supports the property serverName.

```
ds.setServerName("my_database_server");
```

In the following example, the code shows setting properties for the SQLMXDataSource object to use the Type 4 driver to access an SQL/MX database:

```
   SQLMXDataSource ds = new SQLMXDataSource();
   ds.setUrl("jdbc:t4sqlmx://mynode.mycompanynetwork.net:port_number/";
   ds.setCatalog("mycat");
   ds.setSchema("myschema"); ds.setUser("lee");
   ds.setPassword("safeguard password");
// Properties relevant for Type 4 connection pooling.
// Set ds.setMaxPoolSize(-1) for turning OFF connection pooling
ds.setMaxPoolSize("10000");
ds.setMinPoolSize("1000");

// Properties relevant for Type 4 statement pooling.
// Set ds.setMaxStatement(0) for turning statement pooling OFF
// Statement pooling is enabled only when connection pooling is
// enabled.
   ds.setMaxStatements("7000");
```

Programmatically Registering the DataSource Object

In the following example, the code shows how to register, programmatically, the SQLMXDataSource object ds that was created using the preceding code with JDNI.

```
java.util.Hashtable env = new java.util.Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "Factory class name here");
javax.naming.Context ctx = new javax.naming.InitialContext(env);
ctx.rebind("myDataSource", ds);
```

Retrieving a DataSource Instance by using JNDI and to the Data Source

Typically, the JDBC application looks up the data source JNDI name from a context object. After the application has the DataSource object, the application makes a getConnection() call on the data source and gets a connection.

To connect to and use the data source associated with the SQL/MX database, the JDBC application performs the following steps are listed together with the application code to perform the operation:

1. Import the packages.

   ```
   import javax.naming.*;
   import java.sql.*;
   import javax.sql.DataSource;
   ```

2. Create the initial context.

   ```
   Hashtable env = new Hashtable();
   env.put(Context.INITIAL_CONTEXT_FACTORY,
           "com.sun.jndi.fscontext.RefFSContextFactory");
   try
   {
             Context ctx = new InitialContext(env);
   } catch( ... ) { ... }
   ```

3. Look up the JNDI name associated with the data source myDataSource, where myDataSource is the logical name that will be associated with the real world data source – MXCS server.

   ```
   DataSource ds = (DataSource)ctx.lookup("myDataSource");
   ```

4. Using the data source, create the connection.

   ```
   con = ds.getConnection();
   ```

5. Do work with the connection. The following statements are an example:

   ```
   stmt = con.createStatement();
   try
   {
    stmt.executeUpdate("drop table tdata");
   }
   catch (SQLException e) {}
   ```

Specifying the Properties File that Configures the Data Source

To use the properties file method to configure a DataSource object, the properties file must exist on disk and contain the property_name=property_value pairs that configure the data source. See “Creating and Using a Properties File” (page 40) for more information about creating this file.

When the JDBC application makes the connection, the application should pass the properties file as a command-line parameter:

```
java -Dt4sqlmx.properties=file_location ...
```

All the above said properties and configuration are applicable to the SQLMX XADataSource objects participating in distributed transactions.

Connecting using the DriverManager Class

“Loading and Registering the Driver” (page 26)

“Establishing the Connection” (page 26)

“Guidelines for connecting with the Driver Manager” (page 27)

The `java.sql.DriverManager` class is widely used to get a connection, but is less portable than the `DataSource` class. The `DriverManager` class works with the Driver interface to manage the set of drivers loaded. When an application issues a request for a connection using the `DriverManager.getConnection` method and provides a URL, the `DriverManager` finds a suitable driver that recognizes this URL and obtains a database connection using that driver.

`com.tandem.t4jdbc.SQLMXDriver` is the Type 4 driver class that implements the `java.sql.Driver` interface.

Loading and Registering the Driver

Before to the database, the application loads the `Driver` class and registers the Type 4 driver with the `DriverManager` class in one of the following ways:

- Specifies the Type 4 driver class in the `-Djdbc.drivers` option in the command line of the Java program:

  ```
  -Djdbc.drivers=com.tandem.t4jdbc.SQLMXDriver
  ```

- Uses the `Class.forName` method programmatically within the application:

  ```
  Class.forName("com.tandem.t4jdbc.SQLMXDriver")
  ```

- Adds the Type 4 driver class to the `java.lang.System` property `jdbc.drivers` property within the application:

  ```
  jdbc.drivers=com.tandem.t4jdbc.SQLMXDriver
  ```

Establishing the Connection

The `DriverManager.getConnection` method accepts a string containing a Type 4 driver URL. The JDBC URL for the Type 4 driver is:

`jdbc:t4sqlmx://ip_address|machine_name:port_number/[:][property=value [;property2=value2]...]`

where

`ip_address:port_number` specifies the location where the NonStop MXCS association server is running.

`property=value` and *`property2=value2;`* specify a Type 4 driver property name-property value pair. The pairs must be separated by a semicolon (;). For example,

`T4LogLevel=ALL;T4LogFile=temp1.log`

For information about the properties file, see “Type 4 Driver Properties” (page 36).

To establish a connection, the JDBC application can use the following code:

```
Class.forName("com.tandem.t4jdbc.SQLMXDriver"); //loads the driver
String url = “jdbc:t4sqlmx://ip_address|machine_name:port_number/”
Connection con = DriverManager.getConnection(url, “userID”, “Passwd”);
```

The variable `con` represents a connection to the MXCS data source that can be used to create and execute SQL statements.

Guidelines for connecting with the Driver Manager

- The Type 4 driver defines a set of properties that you can use to configure the driver. For more information on these properties, see “Type 4 Driver Properties” (page 36).
- Java applications can specify the properties in the following ways (listed in the order of precedence):
  1. Using the `java.util.Properties` parameter in the `getConnection` method of `DriverManager` class.
  2. Using the database URL in the `DriverManager.getconnection` method, where the URL is:

     `jdbc:t4sqlmx: //ip_addr|machine_name:port_num/property=value`
  3. Using a properties file for the JDBC driver. The properties file is passed as a command-line parameter. The format to enter the properties file in the command-line is:

     `-Dt4sqlmx.properties=location-of-the-properties-file-on-disk`

     For example, `-Dt4sqlmx.properties=C:\temp\t4props`

     For information about the properties file, see “Creating and Using a Properties File” (page 40) in the “Type 4 Driver Properties” (page 36).
  4. Using JDBC properties with the `-D` option in the command line. If used, this option applies to all JDBC connections using the `DriverManager` within the Java application. The format in the command line is:

     `-Dt4sqlmx.property_name=property_value`

     For example, `-Dt4sqlmx.maxStatements=1024`

Stored Procedures

Java applications can use the JDBC standard `CallableStatement` interface to run stored procedures (SPJs) by using the CALL statement. For more information, see the *HPE Nonstop SQL/MX Release 3.2 Guide to Stored Procedures in Java*.

The following SPJ features are not supported:

- `executeBatch` for `callableStatements`
- `set`*`XXX`*`()` methods with parameter names instead of parameter numbers

Starting with SQL/MX Release 3.0, the feature of SPJ returning a `java.sql.ResultSet` is available from T1249V20^ABT.

Connection Pooling

The Type 4 driver provides an implementation of connection pooling, where a cache of physical database connections are assigned to a client session and reused for the database activity. If connection pooling is active, connections are not physically closed. The connection is returned to its connection pool when the `Connection.close()` method is called. The next time a connection is requested by the client, the driver will return the pooled connection, and not a new physical connection.

- The connection pooling feature is available when the JDBC application uses either the `DriverManager` class or `DataSource` interface to obtain a JDBC connection. The

connection pool size is determined by the `maxPoolSize` property value and `minPoolSize` property value.

- By default, connection pooling is disabled. To enable connection pooling, set the `maxPoolSize` property to an integer value greater than 0 (zero).
- Manage connection pooling by using the following Type 4 driver properties:
  - `maxPoolSize`
  - `minPoolSize`
  - `initialPoolSize`
  - `maxStatements`
- When used with the `DriverManager` class, the Type 4 driver has a connection-pool manager that determines which connections are pooled together by a unique value for the following combination of properties:

  ```
  url
  catalog
  schema
  username
  password
  blobTableName
  clobTableName
  serverDataSource
  ```

  Therefore, connections that have the same values for the combination of a set of properties are pooled together.

NOTE: The connection-pooling property values used at the first connection of a given combination are effective throughout the life of the process. An application cannot change any of these property values after the first connection for a given combination.

Statement Pooling

“Guidelines for Statement Pooling” (page 29)

“Troubleshooting Statement Pooling” (page 29)

The statement pooling feature allows applications to reuse the `PreparedStatement` object in same way that they can reuse a connection in the connection pooling environment. Statement pooling is done completely transparent to the application.

Guidelines for Statement Pooling

- To enable statement pooling, set the `maxStatements` property to an integer value greater than 0 and enable connection pooling. See Connection Pooling for more information.
- Enabling statement pooling for your JDBC applications might dramatically improve the performance.
- Explicitly close a `PreparedStatement` by using the `Statement.close` method because `PreparedStatement` objects that are not in scope are also not reused unless the application explicitly closes them.
- To ensure that your application reuses a `PreparedStatement`, call either of the following:
  - `Statement.close` method—called by the application
  - `Connection.close` method—called by the application. All the `PreparedStatement` objects that were in use are ready to be reused when the connection is reused.

Troubleshooting Statement Pooling

Note the following Type 4 driver implementation details if you are troubleshooting statement pooling:

- Type 4 driver looks for a matching `PreparedStatement` object in the statement pool and reuses the `PreparedStatement`. The matching criteria include the SQL string, current catalog, current schema, current transaction isolation, and `resultSetHoldability`. If the Type 4 driver finds the matching `PreparedStatement` object, the driver returns the same `PreparedStatement` object to the application for reuse and marks the `PreparedStatement` object as in use.
- The algorithm, "earlier used are the first to go," is used to make room for caching subsequently generated `PreparedStatement` objects when the number of statements reaches the `maxStatements` limit.
- The Type 4 driver assumes that any SQL CONTROL statements in effect at the time of execution or reuse are the same as those in effect at the time of SQL/MX compilation. If this condition is not true, reuse of a `PreparedStatement` object might result in unexpected behavior.
- You should avoid SQL/MX recompilation to yield performance improvements from statement pooling. The SQL/MX executor automatically recompiles queries when certain conditions are met. Some of these conditions are:
  - A run-time version of a table has a different redefinition timestamp than the compile-time version of the same table.
  - An existing open operation on a table was eliminated by a DDL or SQL utility operation.
  - The transaction isolation level and access mode at execution time is different from that at the compile time.

For more information on SQL/MX recompilation, see the *HPE NonStop SQL/MX Release 3.2.1 Programming Manual for C and COBOL* or the *SQL/MX Programming Manual for Java*.

- When a query is recompiled, the SQL/MX executor stores the recompiled query; therefore, the query is recompiled only once until any of the previous conditions are met again.
- The Type 4 driver pools `CallableStatement` objects in the same way as `PreparedStatement` objects when the statement pooling is activated.
- The Type 4 driver does not cache `Statement` objects.

Thread-safe SQL/MX Access

In the Type 4 driver, API layer classes are implemented as instance-specific objects to ensure thread safety:

- `SQLMXDataSource.getConnection()` is implemented as a synchronized method to ensure thread safety in getting a connection.
- After a connection is made, the connection object is instance-specific.
- If multiple statements are run on different threads in a single connection, statement objects are serialized to prevent data corruption.

"Update ... Where Current of" Operations

When performing an `update ... where current of cursor` SQL statement, the fetch size on a ResultSet must be 1.

If the value of the fetch size is greater than 1, the result of the `update ... where` current of operation might be one of the following:

- An incorrect row might be updated based on the actual cursor position.
- An SQLException might occur, because the cursor being updated might have already been closed.

NOTE: By default, the fetch size is 1.

The following is an example of setting a result set fetch size to 1 and executing an update `... where current of` *`cursor`* SQL statement.

```
ResultSet rs;
... rs.setFetchSize(1);
          String st1 = rs.getCursorName();
          Statement stmt2 = connection.createStatement( ResultSet.TYPE_FORWARD_ONLY
                          , ResultSet.CONCUR_UPDATABLE );
          stmt2.executeUpdate("UPDATE cat2.sch2.table1
                   set j = 'update row' WHERE CURRENT OF " + st1);
```

Internationalization (I18N) Support

String Literals Used in Applications

Internationalization support in the driver affects the handling of string literals. The Type 4 driver handles string literals in two situations:

- The driver processes an SQL statement. For example:

```
Statement stmt = conection.getStatement();
   stmt.execute(“select * from table1 where col1 = ‘abcd’”);
```

- The driver processes JDBC parameters. For example:

```
PreparedStatement pStmt = connection.prepareStatement(“select * from table1 where col1 = ?”);
   pStmt.setString(1, “abcd”);
```

To convert a string literal from the Java character set to an array of bytes for processing by the SQL/MX engine, the Type 4 driver uses the column type and character set in the database. For

information about column data types and character sets, see the *HPE NonStop SQL/MX Release 3.2.1 Reference Manual*.

Controlling String Literal Conversion by Using the Character-Set Properties

The Type 4 driver provides character-set mapping properties. These properties allow you to explicitly define the translation of internal SQL/MX database character-set formats to and from the Java string Unicode (UnicodeBigUnmarked) encoding.

The Type 4 driver provides the following character-set mapping properties using key values as follows:

Key	Default Value
ISO88591	ISO88591_1
KANJI	SJIS
KSC5601	EUC_KR

A description of these character sets appears in the following table, which summarizes the character sets supported by SQL/MX.

Table 2 Corresponding SQL/MX Character Sets and Java Encoding Sets

SQL/MX Character Set	Corresponding JavaEncoding Set—Canonical Name for java.io and java.lang API	Description
ISO88591	ISO88591_1	Single-character, 8- bit, character set for character-data type. ISO88591 supports English and other Western European languages.
KANJI	SJIS	The multi-byte character set widely used on Japanese mainframes. KANJI is composed for a single-byte character set and a double-byte character set. It is a subset of Shift JIS (the double character portion). KANJI encoding is big-endian. **NOTE:** KAJNI is supported in SQL/MP tables only.
KSC5601	EUC_KR	Double-character character set required on systems used by government and banking within Korea. KSC5601 encoding is big-endian. **NOTE:** KSC5601 is supported in SQL/MP tables only.

For more information on these properties, see the following:

- “ISO88591” (page 44)
- “KANJI” (page 45)
- “KSC5601” (page 45)

Using the Character-Set Properties

The `java.sql.PreparedStatement` class contains the methods `setString()` and `setCharacterStream()`. These methods take a String parameter and Reader parameter, respectively.

The `java.sql.ResultSet` class contains the methods `getString()` and `getCharacterStream()`. These methods return a String parameter and Reader parameter, respectively.

Retrieving a Column

When you retrieve a column as a String (for example, call the `getString()` or `getCharacterStream` methods), the Type 4 driver uses the character-set mapping property key to instantiate a `String` object (where that key corresponds to the character set of the column). For example: the following SQL create table statement creates a table that has an ISO88591 column.

```
create table t1 (c1 char(20) character set ISO88591)
```

The JDBC program uses the following `java` command to set the ISO88591 property and issues the `getString()` method.

```
java –Dt4sqlmx.ISO88591=SJIS test1.java
// The following method invocation returns a String object, which
// was created using the “SJIS” Java canonical name as the charset
// parameter to the String constructor.
String s1 = rs.getString(1); // get column 1 as a String
```

Setting a Parameter

When you set a parameter by using a String (for example, call the `setString()` method), the Type 4 driver uses the key's value when generating the internal representation of the String (where that key corresponds to the character set of the column). The character-set parameter to the String `getBytes` method is the Java Canonical name that corresponds to the column's character set.

For example, the following SQL create table statement creates a table that has an ISO88591 column:

```
create table t1 (c1 char(20) character set ISO88591);
> java –DISO88591=SJIS test1.java
```

The following method invocation sets column one of `stmt` to the String “abcd” where “abcd” is encoded as SJIS. The charset parameter to the String `getBytes` method is `SJIS`.

```
stmt.setString(1, “abcd”);
```

Controlling an Exception

You can use the `translationVerification` property to explicitly define the behavior of the driver if the driver cannot translate all or part of an SQL parameter. The value portion of the property can be TRUE or FALSE. (The default value is FALSE).

If the `translationVerification` property's value is FALSE and the driver cannot translate all or part of an SQL statement, the translation is unspecified. In most cases, the characters that are untranslatable are encoded as ISO88591 single-byte question marks ('?' or 0x3F). No exception or warning is thrown.

If the `translationVerification` property's value is TRUE and the driver cannot translate all or part of an SQL statement, the driver throws an SQLException with the following text:

```
Translation of parameter to {0} failed. Cause: {1}
```

where `{0}` is replaced with the target character set and `{1}` is replaced with the cause of the translation failure.

For more information, see the “translationVerification” (page 53).

Trimming Padding for Fixed-Length Character Columns

Retrieved values might be longer than inserted values for fixed-length character columns that use the KANJI character set or KSC5601 character set in SQL/MP tables. This subsection describes:

- “Reinserting a Row that Has Fixed-Length Character Items” (page 33)
- “Discrepancy in Length Caused by Padding Character Causes” (page 33)

Under most circumstances (for example, when the default `Kanji=SJIS` property key-value is set), the padding characters can be inserted and selected from the database without error.

This is an advanced topic on character-set manipulation on SQL/MP tables.

Reinserting a Row that Has Fixed-Length Character Items

NOTE: The length of a retrieved string might not be correct if the associated column is double-byte and fixed length in SQL/MP tables (for example KANJI and KSC5601). The number of padding characters might be more than expected.

To avoid this length problem, ensure that the program trims any trailing white spaces before reinserting such a row before calling the `setString()` method.

Discrepancy in Length Caused by Padding Character Causes

SQL/MP tables containing fixed-length CHAR columns use an ASCII space character (0x20) as a padding character. Double-byte character columns (KANJI and KSC5601) use two ASCII space characters (0x2020) as a padding character. UCS2 uses a single UCS2 space (0x0020) as a padding character.

The 0x2020 character is not a legal character in the Kanji character set, but 0x2020 does represent two legal characters (two spaces) in the SJIS character set. Under most circumstances the padding characters can be inserted and selected from the database without error (for example, when the default `Kanji=SJIS` property key-value is set).

For example, the pseudo code for an insertion into a table that has a column defined as `CHAR(6) character set KANJI`, has the following behavior:

```
byte b1 = new byte[4];
b1[0] = 0x83; \___ Katakana letter A
b1[1] = 0x41; /
b1[2] = 0x83; \___ Katakana letter small i
b1[3] = 0x42; /
String k1 = new String(b1, “SJIS”);
```

Internally, the JVM stores k1 as UCS2 in 4 octets (bytes). The UCS2 encoding would be:

```
0x30 0xA2 0x30 0xA3
```

An SQL insert statement is prepared:

```
pStmt = conn.PrearedStatement(“insert into t1 values (?)”);
```

The statements parameter is set to the string:

```
pStmt.setString(1, k1);
```

Internally, the Type 4 driver creates an array of bytes by using the following pseudo code:

```
byte inB = k1.getBytes(“SJIS”);
int colLen = 12; // i.e. the length of the column (6) times the max length of each character (2)
int padLen = colLen – inB.length; // inB.length = 4 (2 characters times 2 bytes per         character)
inB = inB + 0x20 for padLen bytes;
```

The resulting insert has the following hexadecimal pattern:

```
0x83 0x42 0x83 0x42 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
```

A problem can occur when selecting the inserted row. If you were to select the same row as inserted, and call the `getString()` method, the resulting string would not match the original string. For example:

```
rs = pStmt.executeQuery("Select c1 from t1");
rs.getString(1);
```

Internally, the Type 4 driver creates a string using the following pseudo code:

```
byte[] outB1 = getColumn1(); //
```

where getColumn1() returns the following stored bytes:

```
0x83 0x42 0x83 0x42 0x20 0x20 0x20 0x20 0x20 0x20 0x20 0x20
```

The driver creates a `String` object.

```
String s1 = new String(outB1, "SJIS");
```

The resulting String object has the following UCS2 encoding:

```
0x30 0xA2 0x30 0xA3 0x00 0x20 0x00 0x20 0x00 0x20 0x00 0x20 0x00 0x20 0x00 0x20 0x00 0x20 0x00 0x20
```

The resulting string would be 10 characters (2 characters following by 8 UCS2 spaces). This length is 4 characters more than the column length (which is 6).

Localizing Error Messages and Status Messages

The Type 4 driver supports Internationalization through resource bundles for localized error messages and status messages. The driver uses a set of static strings from a property file to map error messages and status messages to their textual representation.

File-Name Format for the Localized-Messages File

The property file that has the messages must have a file name in the following form:

`SQLMXMessages_`*`xx.properties`*

where *xx* is the locale name. The locale name is defined by the current default locale or by the `language` property.

The Type 4 driver is shipped with an error messages and status messages property file that contains the textual representation of errors and status messages for the English locale. The file is named `SQLMXMessages_en.properties`.

Localized-Message String Format

A localized message file contains strings in the following form:

`message=message_text`

For example:

```
driver_err_error_from_server_msg=An error was returned from the server.
Error: {0} Error detail: {1}
```

where the *`message`* is `driver_err_error_from_server_msg`. The *`message_text`* is: `An error was returned from the server. Error: {0} Errordetail: {1}`

The pattern {*n*} in *`message_text`*, where *n* equals 1, 2, 3, and so forth, is a placeholder that is filled in at run time by the Type 4 driver. Any translation must include these placeholders.

Creating a Localized-Message File

1. Extract the `SQLMXMesseges_en.properties` file, which is in the `t4sqlmx.jar` file. For example from a UNIX prompt, use the `jar` Java tool:

   ```
   jar -x SQLMXMessages_en.properties < t4sqlmx.jar
   ```

2. Copy the file.
3. Edit the file and replace the English text with the text for your locale.

4. Save the file, giving it a file name that meets the naming requirements described under File-Name Format for Localized Messages.
5. Put the file in a directory anywhere in the class path for running the JDBC application.

The new messages file can be anywhere in the class path for running the user application.

At run time, if driver cannot read the messages property file, the driver uses the *message* portion of the property as the text of the message. For a description of the message portion, see the Localized-Message String Format.

At run time, if the Type 4 driver cannot read the messages property file, it sets the locale to English locale, thereby loading the default properties file that is shipped with the Type 4 Driver. The default properties file, `SQLMXMessages_en.properties`, fetches the error messages and status messages.

4 Type 4 Driver Properties

Overview of the Type 4 Driver Properties

Type 4 driver properties that effect client-side operations and server-side are summarized in the following tables. For the detailed description, click the link provided in the property name.

NOTE: Unless otherwise noted in the brief description, the particular property applies to the `DataSource` object, `DriverManager` object, and `ConnectionPoolDataSource` object.

Client-Side Properties

Table 3 Connection Control Properties

Property Name	Description	Default Value
dataSourceName	Specifies the registered DataSource or ConnectionPoolDataSource name. (Can be set only on the DriverManager object.)	None
loginTimeout	Sets the time limit that a connection can be attempted before the connection disconnects.	60 (seconds)
networkTimeout	Sets a time limit that the driver waits for a reply from the database server.	0 (No network timeout is specified.)

Table 4 Pooling Management Properties

Property Name	Description	Default Value
initialPoolSize	Sets the initial connection pool size when connection pooling is used with the Type 4 driver . (Ignored for connections made through the ConnectionPoolDataSource object.)	-1 (Do not create an initial connection pool.)
maxIdleTime	Set the number of seconds that a physical connection can remain unused in the pool before the connection is closed.	0 (Specifies no limit.)
maxPoolSize	Sets the maximum number of physical connections that the pool can contain.	-1 (Disables connection pooling.)
maxStatements	Sets the total number of PreparedStatement objects that the connection pool should cache.	0 (Disables statement pooling.)
minPoolSize	Limits the number of physical connections that can be in the free connection pool.	-1 (The minPoolSize value is ignored.)

Table 5 Operations on CLOB and BLOB Data

Property Name	Description	Default Value
blobTableName	Specifies the LOB table for using BLOB columns.	None
clobTableName	Specifies the LOB table for using CLOB columns.	None
reserveDataLocators	Sets the number of data locators to be reserved for a process that stores data in a LOB table.	100

Table 6 Internationalization Properties

Property Name	Description	Default Value
ISO88591	Sets character-set mapping that corresponds to the SQL/MX ISO88591 character set.	ISO88591_1
KANJI	Sets character-set mapping that corresponds to the SQL/MX KANJI character set.	SJIS (which is shift-JIS, Japanese)
KSC5601	Sets character-set mapping that corresponds to the SQL/MX KSC5601 character set.	ECU_KR (which is KS C 5601,ECU encoding, Korean)
language	Sets the language used for the error messages.	None
translationVerification	Defines the behavior of the driver if the driver cannot translate all or part of an SQL statement or SQL parameter.	FALSE

Table 7 Logging and Tracing Properties

Property Name	Description	Default Value
T4LogFile	Sets the name of the logging file for the Type 4 driver.	The name is defined by the following pattern: `%h/t4sqlmx%u.log`
T4LogLevel	Sets the logging levels that control logging output for the Type 4 driver.	OFF

Table 8 Miscellaneous Client-Side Properties

Property Name	Description	Default Value
description	Specifies the registered source name.	None
properties	Specifies the location of the properties file that contains keyword-value pairs that specify property values for configuring the Type 4 driver .	None

Table 8 Miscellaneous Client-Side Properties *(continued)*

Property Name	Description	Default Value
roundingMode	Specifies the rounding behavior of the Type 4 driver.	ROUND_HALF_UP
useCombinedQueryTimeout	Specifies the behavior of the JDBC driver when Statement.setQueryTimeout() is run.	DEFAULT

Server-Side Properties

Type 4 driver properties that effect server-side operations are summarized in the following tables. Unless otherwise noted in the description, the particular property applies to the `DataSource` object, `DriverManager` object, and `ConnectionPoolDataSource` object.

Table 9 Type 4 Driver Server-Side Properties

Property Name	Description	Default Value
catalog	Sets the default catalog used to access SQL objects referenced in SQL statements if the SQL objects are not fully qualified.	None
connectionTimeout	Sets the number of seconds a connection can be idle before the connection is physically closed by MXCS.	-1 (Use the ConnTimeout value set on the MXCS server data source.)
password	Sets the Guardian password value for to MXCS.	empty string
schema	Sets the database schema that accesses SQL objects referenced in SQL statements if the SQL objects are not fully qualified.	None
serverDataSource	Sets the name of the data source on the MXCS server side (resides on the HPE NonStop server).	None (This value is treated as the default server data source.)
url	Sets the URL value for the MXCS association server. Can be set only on the DriverManager object.	None
useArrayBinding	Improves the performance of SELECT and INSERT statements. It is a server side property.	False
user	Sets the URL value for the MXCS association server.	None

Specifying JDBC Type 4 Properties

“Setting Properties” (page 40)

“Creating and Using a Properties File” (page 40)

“Setting Properties in the Command Line” (page 40)

“Precedence of Property Specifications” (page 40)

The Type 4 JDBC driver properties configure the driver. These properties can be specified in a data source, a connection URL, a properties file, or in the java command line.

Java properties have the form: *key=value*

At runtime, the driver looks for a specific set of property keys and takes action based on their associated values.

Setting Properties

- For connections made through a `DataSource` or a `ConnectionPoolDataSource`, set the property on the `DataSource` or the `ConnectionPoolDataSource` object.
- For the `DriverManager` class, set properties in either of two ways:
  - Using the option `-Dproperty_name=property_value` in the command line
  - Using the `java.util.Properties` parameter in the `getConnection()` method of the `DriverManager` class

Creating and Using a Properties File

JDBC applications can provide property values to configure a connection by using a file that contains properties for the JDBC driver. This property file is passed as a java command-line parameter. The format to enter the properties file in the command line is:

`-Dt4sqlmx.properties=`*`location_of_the_properties_file_on_disk`*

For example,

`-Dt4sqlmx.properties=C:\temp\t4props`

To create the file, use the editor of your choice on your workstation to type in the property values. The entries in properties file must have a property_name=property_value value-pair format:

`property_name=property_value`

For example, `maxStatements=1024`

To configure a DataSource connection, the properties file might contain property names and values as indicated in the following list:

```
url=jdbc:t4sqlmx://mynode.mycompanynetwork.net:port_number/
user=NSK_username
password=mypassword
description=a string
catalog=sampcat
schema=myschema
maxPoolSize=20
minPoolSize=5
maxStatements=20
loginTimeout=15
initialPoolSize=10
connectionTimeout=10
serverDataSource=MXCS_server_data_source
T4LogLevel=OFF
T4LogFile=/mylogdirectory/mylogfile
```

Setting Properties in the Command Line

When a Type 4 driver property is specified on the command line through the `java -D` option, the property must include the prefix:

`t4sqlmx`.

This notation, which includes the period (.), ensures that all the Type 4 driver property names are unique for a Java application. For example, the `maxStatements` property becomes:

`-Dt4sqlmx.maxStatements=10`

Precedence of Property Specifications

If a particular property is set several ways by an application, the value used depends on how the value was set according to the following order of precedence:

1. Set on the `DataSource` object, `DriverManager` object, or `ConnectionPoolDataSource` object.
2. Set through the `java.util.Properties` parameter in the `getConnection` method of `DriverManager` class.
3. Set the property in a properties file specified by the `t4sqlmx.properties` property.
4. Set the `-Dt4sqlmx.`*`property_name`*`=`*`property_value`* in the java command line

For more information, see order of precedence for properties specified in various ways for use with the Driver Manager.

Type 4 Driver Property Descriptions

This section discusses the type 4 driver properties. For information on setting the driver properties, see “Specifying JDBC Type 4 Properties” (page 39).

- “T4LogFile” (page 51)
- “T4LogLevel” (page 52)
- “T4QueryExecuteLogFile” (page 52)
- “translationVerification” (page 53)
- “url” (page 54)
- “useArrayBinding” (page 54)
- “useExternalTransaction” (page 55)
- “user” (page 55)
- autoCommit

The properties are listed in alphabetic order (with the exception of LOB Table Name Properties) together with their descriptions.

blobTableName

See “LOB Table Name Properties” (page 45).

catalog

The catalog property sets the default catalog used to access SQL objects referenced in SQL statements if the SQL objects are not fully qualified.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: none

For example, specifying the catalog `samdcat`:

```
catalog=samdcat
```

clobTableName

See “LOB Table Name Properties” (page 45).

closeConnectionUponQueryTimeout

The closeConnectionUponQueryTimeout property specifies the behavior of the JDBC driver when Statement.setQueryTimeout() is run.

The values that can be provided for the property closeConnectionUponQueryTimeout are IGNORE/DEFAULT/ON.

If the property value is set to any value other than the ones mentioned, the value is set to DEFAULT.

The JDBC Driver behaves in accordance to the set value. Listed below are the effects on the JDBC Driver when the IGNORE, DEFAULT, or ON value is set:

IGNORE = Any value (> 0) set by calling Statement.setQueryTimeout() has no effect. The Statement continues to block the current thread until the statement is run.

DEFAULT = This is the value set if the property is not specified. Any value (> 0) set by calling Statement.setQueryTimeout() causes anSQLException to be raised when this property is set to DEFAULT.

ON = Any value (>0) set by Statement.setQueryTimeout() has the following effect:

If the Statement takes more time than the specified timeout value, the current Connection object is terminated and an SQLException is raised.

For example, specify the value IGNORE as follows:

```
closeConnectionUponQueryTimeout=IGNORE
```

connectionTimeout

The `connectionTimeout` property sets the number of seconds a connection can be idle before the connection is physically closed by MXCS.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: int

Units: seconds

Default: -1 (Use the `ConnTimeout` value set on the MXCS server data source.)

Range: -1, 0 to 2147483647

- Zero (0) specifies infinity as the timeout value.
- A non-zero positive value overrides the value set on the MXCS server data source, if allowed by the MXCS settings. For more information, see the *HPE NonStop Connectivity Server Manual for SQL/MX Release 3.2.1*.
- A negative value is treated as -1.

For an example, consider this scenario. Even if a connection is not being used, it takes up resources. The application abandons connections; that is, the application does not physically close a connection after the application finishes using the connection. However, you can configure the connection to close itself after 300 seconds by setting the `connectionTimeout` property. Then, when a connection is not referenced for 300 seconds, the connection automatically closes itself. In this example, the specification to set the `connectionTimeout` property is:

```
connectionTimeout=300
```

dataSourceName

The `dataSourceName` property specifies the registered `DataSource` or `ConnectionPoolDataSource` name.

Set this property on the `DataSource` object.

Data type: String

Default: none

For example: `dataSourceName=myDataSource`

description

The `description` property specifies the registered source name.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: none

The value can be any valid identifier.

executeBatchWithRowsAffected

The `executeBatchWithRowsAffected` property changes the behavior of `java.sql.statement.executeBatch()`. If the property is set to ON, the `java.sql.statement.executeBatch()` returns an array that has the number of rowsaffected

value for each `java.sql.statement.addBatch()`. If a SQL operation fails, the entire batch operation rolls back. Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: OFF

Values : ON or OFF

For example: `executeBatchWithRowsAffected=ON`

initialPoolSize

The `initialPoolSize` property sets the initial connection pool size when connection pooling is used with the Type 4 driver.

Set this property on a `DataSource` object or `DriverManager` object. This property is ignored for connections made through the `ConnectionPoolDataSource` object. The driver creates *n* connections (where *n* is `initialPoolSize`) for each connection pool when the first connection is requested. For example, if `initialPoolSize` is set to 5 for a data source, the driver attempts to create and pool five connections the first time the application calls the data source's `getConnection()` method.

Data type: int

Units: number of physical connections

Default: -1 (Do not create an initial connection pool.)

Range: -1 to `maxPoolSize`

- Any negative value is treated as -1.
- Values can be less than `minPoolSize`, but must not exceed `maxPoolSize`. If the specified value is greater than `maxPoolSize`, the `maxPoolSize` property value is used.

For example: `initialPoolSize=10`

ISO88591

The ISO88591 character-set mapping property corresponds to the SQL/MX ISO88591 character set, which is a single- byte 8-bit character set for character data types. This property supports English and other Western European languages. For more information, see Internationalization (I18N) Support.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: `ISO88591_1`

The value can be any valid Java Canonical Name as listed in the "Canonical Name for java.io and java.lang API" column of the Oracle documentation, Supported Encodings .

fetchBufferSize

The fetchBufferSize property limits the number of kilobytes to fetch from the server.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For more information, see

"Specifying JDBC Type 4 Properties" (page 39).

Data type: int

Default: 512

Range: 0 through 32767

Zero and negative values are treated as default values.

KANJI

The KANJI character-set mapping property corresponds to the SQL/MX KANJI character set, which is a double-byte character set widely used on Japanese mainframes. This property is a subset of Shift JIS -- the double character portion. The encoding for this property is bigendian. Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: `SJIS` (which is shift-JIS, Japanese)

For example, `java –Dt4sqlmx.KANJI=SJIS`

For more information, see “Internationalization (I18N) Support” (page 30).

KSC5601

The KSC5601 character-set mapping property corresponds to the SQL/MX KSC5601 character set, which is a double-byte character set. Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: `ECU_KR` (which is KS C 5601, ECU encoding, Korean)

The value can be any valid Java Canonical Name as listed in the "Canonical Name for java.io and java.lang API" column of the Oracle documentation, Supported Encodings.

For example:

```
java –Dt4sqlmx.KSC5601=ECU_KR
```

For more information, see “Internationalization (I18N) Support” (page 30).

language

The language property sets the language used for the error messages. For more information about using this property, see Localizing Error and Status Messages.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: none

The value can be any valid Java Canonical Name as listed in the "Canonical Name for java.io and java.lang API" column of the Oracle documentation, Supported Encodings.

For example, to set the language to shift-JIS, Japanese:

```
language=SJIS
```

LOB Table Name Properties

LOB tables store data for `BLOB` columns and `CLOB` columns. The following properties specify the LOB table for using `BLOB` columns or `CLOB` columns:

- For the binary data for `BLOB` columns

  `blobTableName`

- For the character data for `CLOB` columns

  `clobTableName`

The property value is of the form:

catalog_name.schema_name.lob_table_name

Data type: String

Default: none

For example:

`blobTableName=samdbcat.sales.lobvideo`

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

loginTimeout

The `loginTimeout` property sets the time limit that a connection can be attempted before the connection disconnects. When a connection is attempted for a period longer than the set value, in seconds, the connection disconnects.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: int

Units: seconds

Default: 60

Range: 0 to `java.lang.Short.MAX_VALUE` (32767)

If set to 0 (zero), no login timeout is specified. If set to a value more than the specified range such as 330000, the following error message is displayed:

```
Invalid connection property setting: Provided LoginTimeout property
value 33000 is invalid and should be less than Short.MAX_VALUE(32767.
```

maxIdleTime

NOTE: The `maxIdleTime` property is available in the V11^AAC and subsequent Software Product Releases (SPRs).

The `maxIdleTime` property determines the number of seconds that a physical connection must remain unused in the pool before the connection is closed. 0 (zero) indicates no limit.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: int

Units: seconds

Default: 0 (No timeout)

Range: 0 through 2147483647

Any negative value is treated as 0, which indicates that no time limit applies.

For example, to set the maximum idle time to 5 minutes (300 seconds):

```
java –Dt4sqlmx.maxIdleTime=300
```

maxPoolSize

The maxPoolSize property sets the maximum number of physical connections that the pool can contain. These connections include both free connections and connections in use. When the maximum number of physical connections is reached, the Type 4 driver throws an SQLException and sends the message, `Maximum pool size is reached`.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: int

Units: number of physical connections

Default: -1 (Disables connection pooling.)

Range: -1, 0 through 2147483647, but greater than `minPoolSize`

The value determines connection-pool use as follows:

Any negative value is treated like -1.

0 means no maximum pool size.

A value of -1 disables connection pooling.

Any positive value less than `minPoolSize` is changed to the `minPoolSize` value.

maxStatements

The `maxStatements` property sets the total number of `PreparedStatement` objects that the connection pool must cache. This total includes both free objects and objects in use. The JDBC T4 driver side cache uses the Most Frequently Used (MFU) or the Most Recently Used (MRU) algorithms. MFU is the default setting. For more information on selecting the algorithm that you want, see “mfuStatementCache” (page 47).

MFU-When the number of `PreparedStatements` in the cache exceeds the specified `maxStatements` property, the least used `PreparedStatement` is closed and removed from the statement cache to allow a new statement.

MRU-When the number of `PreparedStatements` in the cache exceeds the specified `maxStatements` property value, the recently used `PreparedStatements` are retained and the earliest `PreparedStatement` is removed from the statement cache.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: int

Units: number of objects

Default: 0 (Disables statement pooling.)

Range: 0 through 2147483647

The value 0 disables statement pooling. Any negative value is treated like 0 (zero).

Performance tip:

Hewlett Packard Enterprise recommends that you enable statement pooling for your JDBC applications because this pooling helps the performance of many applications.

For example, to specify statement pooling, type: `maxStatements=10`

The working of the `PreparedStatement` Caching algorithm is as described in the following section.

mfuStatementCache

The JDBC T4 driver supports the following statement caching algorithms:

- Most Frequently Used (MFU)
- Most Recently Used (MRU)

The `mfuStatementCache` property enables you to select a statement caching algorithm. MFU is the default caching algorithm.

Set this property on a DataSource object, ConnectionPoolDataSource object, or DriverManager object.

Data type: String

Default: YES

Available values: YES and NO. For example, mfuStatementCache = NO

For information on enabling caching, see “maxStatements” (page 47).

minPoolSize

The minPoolSize property limits the number of physical connections that can be in the free connection pool.

Set this property on a DataSource object, ConnectionPoolDataSource object, or DriverManager object.

Data type: int

Default: -1 (The minPoolSize value is ignored.)

Range: -1, 0 through n, but less than maxPoolSize

Any negative value is treated like -1. Any value greater than maxPoolSize is changed to the maxPoolSize value. The value of minPoolSize is set to -1 when maxPoolSize is -1. The value determines connection pool use as follows:

- When the number of physical connections in the free pool reaches the minPoolSize value, the Type 4 driver closes subsequent connections by physically closing them—and not adding them to the free pool.
- 0 (zero) means that the connections are not physically closed; the connections are always added to the free pool when the connection is closed.

For example, use the following specification to set the minPoolSize value to 1, which ensures that one connection is always retained:

```
minPoolSize=1
```

networkTimeout

The networkTimeout property sets a time limit that the driver waits for a reply from the database server. When an operation is attempted for a period longer than the set value, in seconds, the driver stops waiting for a reply and returns an SQLException to the user application.

Set this property on a DataSource object, ConnectionPoolDataSource object, or DriverManager object.

NOTE: Be careful when using this property. A network timeout causes the socket connection between the Type 4 driver and the MXCS server to timeout. If the server is engaged in a transaction or an SQL operation, then the server continues to perform that transaction or operation until the transaction or operation fails, TMF times out, or the server realizes that the Type 4 driver client has gone away. A network timeout can result in an open transaction or operation that continues for a significant time before failing or rolling back.

As a result of a network timeout, the connection becomes unavailable.

Data type: int

Units: seconds

Default: 0 (No network timeout is specified.)

0 through to 2147483647

password

The `password` property sets the Guardian password value for to MXCS. The password is encrypted when it is passed to MXCS. Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: empty string

For example: `password=eye0weU$s`

properties

The `properties` property specifies the location of the properties file that contains keyword-value pairs that specify property values for configuring the Type 4 driver. For more information, see “Creating and Using a Properties File” (page 40).

queryExecuteTime

This property is used to specify the value in time (milliseconds). Any query that takes more than the specified time will be logged in the log file. If this property is specified, a separate log file is created. The default log file name is `t4sqlmxQueryExecuteTime.log` and will be created in the `user.home` directory.

If you want to explicitly specify a log file, see the “T4QueryExecuteLogFile” (page 52).

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or a `DriverManager` object.

Data type: int

Units: Number of milliseconds

Default: None

Range: 1 to 9,223,372,036,854,775,807 (2**63-1)

For example: `java -Dt4sqlmx.queryExecuteTime = 10`

reserveDataLocators

The `reserveDataLocators` property sets the number of data locators to be reserved for a process that stores data in a LOB table. Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: int

Units: number of data locators to be reserved

Default: 100

Range: 1 to 9,223,372,036,854,775,807 (2**63 -1)

Do not set a value much greater than the number of data locators actually needed. If the specified value is 0 (zero) or less, the default value (100) is used.

Base the setting of the value of the `reserveDataLocators` property on the application profile being run. If the application inserts a large number of LOB items, a higher value of the `reserveDataLocators` property can prevent frequent updating of the ZZ_DATA_LOCATOR value in the LOB table. However, if the application inserts only a small number of LOB items, a smaller value is better. If a large value is used, holes (unused data-locator numbers) might occur in the LOB table.

Also, the administrator should avoid setting high values for the `reserveDataLocators` (for example in the range of trillions or so). Setting high values prevents other Type 4 applications that use LOB table from reserving data locators.

For additional information about data locator use, see “Reserving Data Locators” (page 59).

To change this value for a JDBC application, specify this property from the command line. For example, the following command reserves 150 data locators for program class `myProgramClass`.

```
java –Dt4sqlmx.reserveDataLocators=150 myProgramClass
```

roundingMode

The `roundingMode` property specifies the rounding behavior of the Type 4 driver. For example, if the data is 1234.127 and column definition is numeric(6, 2) and the application does `setDouble()` and `getDouble()`, the value returned is 1234.12, which is truncated as specified by the default rounding mode, `ROUND_HALF_UP`.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: `ROUND_HALF_UP`

Values for `roundingMode` are:

```
ROUND_CEILING
ROUND_DOWN
ROUND_FLOOR
ROUND_HALF_DOWN
ROUND_HALF_EVEN
ROUND_HALF_UP
ROUND_UNNECESSARY
ROUND_UP
```

- For the definition of rounding mode values, see the `java.math.BigDecimal` documentation in http://download.oracle.com/javase/1.5.0/docs/api/index.html pages.
- If the application sets erroneous values for the `roundingMode` property, no error is thrown by the Type 4 driver. The driver uses `ROUND_HALF_UP` value instead.
- To have the application get the DataTruncation exception when data is truncated, set the roundingMode property to `ROUND_UNNECESSARY`.
- The default rounding mode is `ROUND_HALF_UP`.

schema

The `schema` property sets the database schema that accesses SQL objects referenced in SQL statements if the SQL objects are not fully qualified.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: none

For example, `schema=sales`

serverDataSource

The `serverDataSource` property sets the name of the data source on the MXCS server side (resides on the HPE NonStop server).

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

The `serverDataSource` allows the application to set SQL/MX properties (such as resource governing) for server-side data sources. For more information about server-side data sources, see "Server Data Sources" in the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

- If not specified, the default server data source is used.
- If the data source named in `serverDataSource` is not available on the MXCS server, the default server data source is used.
- If the data source named in `serverDataSource` is not started on the MXCS server, an `SQLException` is thrown.

  Data type: String

  Default: `None` (This value is treated as the default server data source.)

For example, the MXCS server default `serverDataSource` value is:

```
serverDataSource=MyDataSource
```

For more information about the default server data source, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

T4LogFile

The `T4LogFile` property sets the name of the logging file for the Type 4 driver.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default file name is defined by the following pattern:

`%h/t4sqlmx%u.log`

where

`/` represents the local pathname separator.

%h represents the value of the `user.home` system property.

%u represents a unique number to resolve conflicts.

Any valid file name for your system is allowed.

If you explicitly specify a log file, that file is overwritten each time a `FileHandler` is established using that file name.

To retain previously created log files, use the standard `java.util.logging` file syntax to append a unique number onto each log file. For example, you can have the following property in a data source:

```
T4LogFile = C:/temp/MyLogFile%u.log
```

That name causes the Type 4 driver to create a new log file using a unique name for each connection made through that data source. For example:

```
C:/temp/MyLogFile43289.log
C:/temp/MyLogFile87634.log
C:/temp/MyLogFile27794.log
```

If you explicitly specify a log file that is not fully qualified, the Type 4 driver creates the file in the current working directory, for example, in the directory from which the JVM was invoked.

For detailed information about `java.util.logging`, see the logging summary at:

http://download.oracle.com/javase/1.5.0/docs/api/java/util/logging/package-summary.html

T4LogLevel

The T4LogLevel property sets the logging levels that control logging output for the Type 4 driver. The Java package java.util.logging logs error messages and traces messages in the driver.

Set this property on a DataSource object, ConnectionPoolDataSource object, or DriverManager object.

Data type: String

Default: OFF

Logging Levels

OFF	is a special level that turns off logging; the default setting.
SEVERE	indicates a serious failure; usually applies to SQL exceptions generated by the Type 4 driver.
WARNING	indicates a potential problem, which usually applies to SQL warnings generated by the Type 4 driver.
INFO	provides informational messages, typically about connection pooling, statement pooling, and resource usage. This information can help in tuning application performance.
CONFIG	provides static configuration messages that can include property values and other Type 4 driver configuration information.
FINE	provides tracing information from the Type 4 driver methods described in the Type 4 driver API. The level of tracing is equivalent to the level of tracing provided when calling the setLogWriter() method of the DriverManager class or the DataSource class.
FINER	indicates a detailed tracing message for which internal Type 4 driver methods provide messages. These messages can be useful in debugging the Type 4 driver.
FINEST	indicates a highly detailed tracing message. The driver provides detailed internal data messages that can be useful in debugging the Type 4 driver.
ALL	logs all messages.

For example, to enable tracing, use the t4sqlmx.T4LogLevel property specified in the command line:

```
-Dt4sqlmx.T4LogLevel=FINE
```

T4LogLevel Considerations

- If a security manager is defined by your application using an AppServer, LoggingPermission must be granted in the java.policy file as follows:

  ```
  permission java.util.logging.LoggingPermission "control","";
  ```

- The Type 4 driver is not designed to inherit the java.util.logging.FileHandler.level settings at program startup.
- Server-side tracing and logging through MXCS is managed by NSM/web. For more information about server side-tracing (logging), see the NSM/web online help.

T4QueryExecuteLogFile

Whenever the queryExecuteTime is specified, T4QueryExecuteLogFile property is used to set the name of the logging file for the queryExecuteTime property.

Set this property on a DataSource object, ConnectionPoolDataSource object, or a DriverManager object.

Data type: String

The default log file name is `t4sqlmxQueryExecuteTime.log` and will be created in the `user.home` directory.

The system can accept any valid file name. If a log file is specified explicitly, it is overwritten each time a FileHandler is established using that filename.

If a log file that is not fully qualified is specified explicitly, the Type 4 driver creates the file in the current working directory, for example, in the directory from where the JVM was invoked.

sslEncryption

Starting with H06.27 and J06.16 RVUs, the Type 4 driver supports Secure Sockets Layer (SSL) encryption for communicating with the ODBC/MX server. SSL and its successor Transport Layer Security (TLS) are cryptographic protocols that provide communication security over the network. The Type 4 driver uses the standard Java package (`javax.net.ssl`) for SSL support. You can configure SSL encryption using the runtime JVM property, `sslEncryption`.

To enable SSL encryption, set the `sslEncryption` property to ON. You can use the command line and set the property as follows:

```
-Dt4sqlmx.sslEncryption = ON
```

You can also set this as a system property in the application using the following method:

```
system.setproperty("t4sqlmx.sslEncryption", "ON")
```

NOTE: You cannot set this property on a `DataSource` object or in the properties file.

Data type: String

The default value is OFF.

NOTE: You must add the server certificate to the client trusted certificates list. Java provides a built-in list of trusted certificates, and the `keytool` utility to add a new certificate to the trust store. You can use the following command to add the certificate to the default Java trust store:

```
keytool –import –alias <aliasname> –keystore
<jdk-install-location>\jre\lib\security\cacerts
–file <server CA certificate>
```

You can create a new trust store and use this store at runtime with the following commands:

```
keytool –import –alias <aliasname> –keystore <truststorename>.jks
–file <server CA certificate>
-Djavax.net.ssl.trustStore=<truststorename>.jks
-Djavax.net.ssl.trustStorePassword=<password>
```

translationVerification

The `translationVerification` property defines the behavior of the driver if the driver cannot translate all or part of an SQL statement or SQL parameter.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

The value can be `TRUE` or `FALSE`.

Data type: String

Default: `FALSE`

If the `translationVerification` property's value is `FALSE`, and the driver is unable to translate all or part of an SQL statement; the translation is unspecified. In most cases, the

characters that are untranslatable are encoded as ISO88591 single-byte question marks (? or 0x3F). No exception or warning is thrown.

If the translationVerification property's value is TRUE and the driver cannot translation all or part of an SQL statement or parameter, the driver throws an SQLException with the following text.

Translation of parameter to *{0}* failed. Cause: *{1}*

where *{0}* is replaced with the target character set and *{1}* is replaced with the cause of the translation failure.

If the translationVerification property is set to TRUE, the process can use significantly more system resources. For better performance, set this property to FALSE.

For more information, see "Internationalization (I18N) Support" (page 30).

url

The url property sets the URL value for the MXCS association server. This property is used in the DriverManager object. The format to specify the URL is:

```
jdbc:t4sqlmx//ip_address|machine_name:port_number/[:]
      [property=value[;property2=value]...]
```

where *ip_address*|*machine_name:port_number* specifies the location where the MXCS association server is running.

Data type: String

Default: none

For example:

```
url=jdbc:t4sqlmx://mynode.mycompanynetwork.net:18000/
```

url Property Considerations

- If the url parameter is not specified and DriverManager.getConnection() is called, the Type 4 driver throws an SQLException.
- If you use a literal IPV4 or IPV6 address in a URL, note the following guidelines:
  - For IPV6 only, enclose the address in brackets ([and]).
  - The port number is optional according to both the IPV4 and IPV6 standard.
  - The default port number for the MXCS association server is 18650.

useArrayBinding

useArrayBinding property improves the performance of the SELECT and INSERT statements.

When this property is set to "true", the JDBC T4 driver sends/receives multiple rows to/from the MXOSRVR, which in turn sends/receives information about multiple rows to/from SQL/MX.

This property is ideal for bulk SELECT and INSERT operations.

useArrayBinding property has no influence on UPDATE and DELETE statements.

This property accepts only the following values:

true

false

Data Type: String

Default: false

For example, specify the value `true` as

`useArrayBinding = true`

useExternalTransaction

This property is used to inherit the external applications transaction running on the NonStop system. If the transaction exists, the database operation is performed within the specified transaction. If the property is set and an external transaction does not exist or is not passed, you cannot perform database-related operations. As a result, exceptions are generated.

The following is the exception:

`Invaild/Nil transaction handle,useExternalTransaction property enabled`

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or a `DriverManager` object.

Data type: String

Default Value: NO

Values: NO or YES

For example: `useExternalTransaction = YES`

Considerations:

- This feature is supported only when the client side JVM (loading the Type 4 driver) is running on the NonStop platform (when the Type 4 driver is used instead of the Type 2 driver).
- This feature requires NS Java 1.6, SPR T2766^ACA or later.
- `Java.sql.connection.setAutoCommit(), Java.sql.connection.commit(),` and `Java.sql.connection.rollback()` methods are **NO-OP** when this property is enabled.
- BLOB, CLOB, and XA features are not supported when this property is enabled.
- `executeBatchWithRowsAffected` is not supported.

user

The `user` property sets the Guardian user-name value for to MXCS. The Guardian user name passed must have adequate access permissions for SQL/MX data accessed through MXCS.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object.

Data type: String

Default: empty string

For example, `user=NonStopSystem_username`

autoCommit

The `autoCommit` property specifies whether SQL/MX automatically commits or rolls back if an error occurs at the end of a statement execution. This property applies to any statement for which the system initiates a transaction. If this property is set to `true`, SQL/MX automatically commits any changes or rolls back any changes made to the database at the end of a statement execution.

By default, this property is set `true` when a JDBC connection is established. If `autoCommit` property is set to `false`, the current transaction remains active until the application explicitly commits or rolls back the transaction. If the JDBC/MX Type 4 driver property `t4sqlmx.autoCommit` is specified, and JDBC API method `java.sql.Connection.setAutoCommit()` is called from the application, the values specified in the API method take higher precedence over the property `t4sqlmx.autoCommit`.

Valid values: `ON` or `OFF`

Default value: `ON`

Data type: `String`

For example, `-Dt4sqlmx.autoCommit = OFF`

Example 1 Example for connection setting made through a `DataSource` or a `ConnectionPoolDataSource`

```
SQLMXDataSource ds = new SQLMXDataSource();
  ds.setAutoCommit("OFF");
  try {
   Connection Conn = ds.getConnection();
  } catch (SQLException e) {
   e.printStackTrace();
```

Example 2 Example for setting using a properties file

```
maxStatements=20
loginTimeout=15
initialPoolSize=10
connectionTimeout=10
serverDataSource=MXCS_server_data_source
autoCommit=OFF
```

Example 3 Example for setting using command line

```
-Dt4sqlmx.autCommit=OFF
```

Considerations:

When connection pooling is enabled, the driver applies the `autoCommit` setting on the connection object before returning the object to the application.

5 Working with BLOB and CLOB Data

This chapter describes working with BLOB and CLOB data in JDBC applications. You can use the standard interface described in the JDBC 3.0 API specification to access BLOB and CLOB data in NonStop SQL/MX tables with support provided by the Type 4 driver.

BLOB and CLOB are not native data types in an SQL/MX database. But, database administrators can create SQL/MX tables that have BLOB and CLOB columns by using the Type 4 driver or special SQL syntax in MXCI as described under Creating Base Tables that Have LOB Columns.

For management purposes, CLOB and BLOB data is referred to as large object (LOB) data, which can represent either data type.

NOTE: CLOB data types are not supported when using stored procedures in Java (SPJs).

For information about creating and managing tables for BLOB and CLOB data, see “Managing the SQL/MX Tables for BLOB and CLOB Data” (page 67).

Architecture for LOB Support

The tables that support LOB data are:

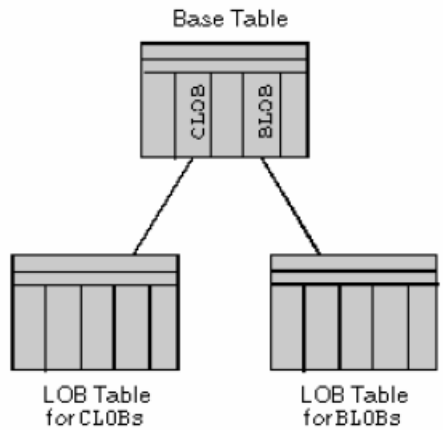
Base table

Referenced by JDBC applications to insert, store, read, and update BLOB and CLOB data. In the base table, the Type 4 driver maps the BLOB and CLOB columns into a data-locator column. The data-locator column points to the actual LOB data that is stored in a separate user table called the LOB table.

LOB table

Actually contains the BLOB and CLOB data in chunks. A Clob or Blob object is identified by a data locator. LOB tables have two formats: LOB table for BLOB data and a LOB table for CLOB data.

Figure 3 LOB Architecture: Tables for LOB Data Support



Types of LOB Table

The LOB table is a separate table from the base table that stores the LOB data. The two types of LOB table are: Binary or ASCII LOB and Unicode LOB.

The Binary or ASCII LOB table structure is:

```
table_name CHAR(128) NOT NULL NOT DROPPABLE,
data_locator LARGEINT NOT NULL NOT DROPPABLE,
chunk_no INT NOT NULL NOT DROPPABLE,
lob_data VARCHAR(3880)
```

The Unicode LOB table structure for ASCII data is:

```
table_name CHAR(128),
data_locator LARGEINT,
chunk_no INTEGER,
lob_data VARCHAR(1955) CHARACTER SET UCS2
```

Setting Properties for the LOB Table

Before running the JDBC application that uses BLOB and CLOB data through the JDBC API, the database administrator must create the LOB tables. For information on creating LOB tables, see Managing LOB Data with the Lob Admin Utility.

The JDBC applications that access BLOB or CLOB data must specify the associated LOB table names and, optionally, configure the `reserveDataLocators` property.

Specifying the LOB Table

At run time, a user JDBC application notifies the Type 4 driver of the name or names of the LOB tables associated with the `CLOB` or `BLOB` columns of the base tables being accessed by the application. One LOB table or separate tables can be used for `BLOB` and `CLOB` data. The JDBC application specifies a LOB table name either through a system parameter or through a Java `Property` object by using one of the following properties, depending on the LOB column type:

LOB Column Type	Property name
`BLOB`	`blobTableName`
`CLOB`	`clobTableName`

For more information about using these properties, see "LOB Table Name Properties" (page 45).

Reserving Data Locators

A data locator is the reference pointer value (SQL LARGEINT data type) that is substituted for the `BLOB` or `CLOB` column in the base table definition. Each object stored into the LOB table is assigned a unique data locator value. Because the LOB table is a shared resource among all accessors that use the particular LOB table, reserving data locators reduces contention for getting the next value. The default setting is 100 reserved data locators; therefore, each JVM instance can insert 100 large objects (not chunks) before needing a new allocation.

Specify the number of data locators (`n`) to reserve for your application by using the Type 4 driver property `t4sqlmx.reserveDataLocators`. For information about specifying this property, see "reserveDataLocators" (page 49).

Storing CLOB Data

Inserting CLOB Columns by Using the Clob Interface

When you insert a row containing a CLOB data type, and before the column can be updated with real CLOB data, you can insert a row that has an empty CLOB value. To insert an empty CLOB value in a NonStop SQL/MX database, specify the EMPTY_CLOB( ) function for the CLOB column in the insert statement.

NOTE:

- The EMPTY_CLOB( ) function is an HPE NonStop specific function and might not work on other databases.
- Do not use the EMPTY_CLOB( ) function when using the PreparedStatement interface.

The Type 4 driver scans the SQL string for the EMPTY_CLOB( ) function and substitutes the next-available data locator.

Then, obtain the handle to the empty CLOB column by selecting the CLOB column for update. The following code illustrates how to obtain the handle to an empty CLOB column:

```
Clob myClob = null;
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("Select myClobColumn
  from myTable where ….for update");
if (rs.next())
  myCLOB = rs.getClob(1);
```

You can now write data to the CLOB column. See “Writing ASCII or MBCS Data to a CLOB Column” (page 60) or “Writing Unicode data to a CLOB column” (page 61).

NOTE: Limitation: Do not rename the CLOB column in the select query.

Writing ASCII or MBCS Data to a CLOB Column

You can write ASCII or MBCS data to a CLOB column.

NOTE: Multibyte Character Set (MBCS) data and ASCII data are handled the same way.

ASCII Data

To write ASCII or MBCS data to the CLOB column, use the Clob interface. The following code illustrates using the setAsciiStream method of the Clob interface to write CLOB data.

```
// stream begins at position: 1
long pos = 1;
//String containing the ASCII data
String s ;
// Obtain the output stream to write Clob data
OutputStream os = myClob.setAsciiStream(pos);
// write Clob data using OutputStream
byte[] myClobData = s.getBytes();
os.write(myClobData);
```

The Type 4 driver splits the output stream into chunks and stores the chunks in the LOB table.

Inserting CLOB Data by Using the PreparedStatement Interface

You can use the PreparedStatement interface to insert a CLOB column using ASCII or MBCS data.

ASCII Data

To insert a CLOB column using ASCII or MBCS data from an InputStream, use the PreparedStatement interface to insert the CLOB column.

```
InputStream inputAsciiStream;
PreparedStatement ps = conn.prepareStatement("insert
    into myTable (myClobColumn) values (?)");
ps.setAsciiStream(1, inputAsciiStream, length_of_data);
ps.executeUpdate();
```

The Type 4 driver reads the data from `InputStream` and writes the data to the LOB table. The Type 4 driver substitutes the next-available data locator for the parameter of the `CLOB` column in the table.

Inserting a `Clob` Object by Using the `setClob` Method

Your JDBC application cannot directly instantiate a `Clob` object. To perform an equivalent operation:

1. Obtain a `Clob` object by using the `getClob` method of the `ResultSet` interface.
2. Insert the `Clob` object into another row by using the `setClob` method of the `PreparedStatement` interface.

In this situation, the Type 4 driver generates a new data locator and, when the `PreparedStatement` is executed, copies the contents of the source `Clob` into the new `Clob` object.

Inserting a `CLOB` column with Unicode data using a Reader

You can use the `PreparedStatement` interface to insert a `CLOB` column with Unicode data using a Reader.

```
Reader inputReader;
PreparedStatement ps = conn.prepareStatement("insert into
myTable (myClobColumn) values (?)");
ps.setCharacterStream(1, inputReader, length_of_data);
ps.executeUpdate();
```

Type 4 driver reads the data from a Reader and internally `SQLMXClobWriter` writes the data to the LOB table. Type 4 driver substitutes the next available data locator to the parameter of the CLOB column in the base table.

Writing Unicode data to a `CLOB` column

The following code illustrates how to write a Unicode data to a CLOB, after obtaining the handle to the empty CLOB column:

```
long pos = 0;
// String containing the Unicode data
String s ;
// Obtain the output stream to write Clob data
Writer cw = myClob.setCharacterStream(pos);
// write Clob data using Writer
char[] myClobData = s.toCharArray();
cw.write(myClobData);
```

Reading CLOB Data

Reading ASCII Data from a `CLOB` Column

To read ASCII or MBCS data from a `CLOB` column, use the `Clob` interface or `InputStream`.

Using the `Clob` interface:

```
// Obtain the Clob from ResultSet
Clob myClob = rs.getClob("myClobColumn");
```

```
// Obtain the input stream to read Clob data
InputStream is = myClob.getAsciiStream();
// read Clob data using the InputStream
byte[] myClobData;
myClobData = new byte[length];
is.read(myClobData, offset, length);
```

To read ASCII or MBCS data from the CLOB column by using InputStream:

```
// obtain the InputStream from ResultSet
InputStream is = rs.getAsciiStream("myClobColumn");
// read Clob data using the InputStream
byte[] myClobData;
myClobData = new byte[length];
is.read(myClobData, offset, length);
```

Reading Unicode data from a CLOB Column

To read Unicode or MBCS data from a CLOB column, use the CLOB interface or Reader.

Using the CLOB interface:

```
// Obtain the Clob from ResultSetClob
myClob = rs.getClob("myClobColumn");
// Obtain the input stream to read Clob data
Reader cs = myClob.getCharacterStream();
// read Clob data using Reader
char[] myClobData;
myClobData = new char[length];
cs.read(myClobData, offset, length);
```

To read Unicode data from a CLOB column by using a Reader:

```
// obtain the Reader from ResultSet
Reader cs = rs.getCharacterStream("myClobColumn");
// read Clob data using the InputStream
char[] myClobData;
myClobData = new char[length];
css.read(myClobData, offset, length);
```

Updating CLOB Data

To update CLOB data, use the methods in the Clob interface or use the updateClob method of the ResultSet interface. The Type 4 driver makes changes directly to the CLOB data. Therefore, the Type 4 driver returns false to the locatorsUpdateCopy method of the DatabaseMetaData interface. Applications do not need to issue a separate update statement to update the CLOB data.

Make updates to CLOB data in the following ways:

- “Updating Clob Objects with the updateClob Method” (page 62)
- “Replacing Clob Objects” (page 63)

Updating Clob Objects with the updateClob Method

Unlike some LOB support implementations, the Type 4 driver updates the CLOB data directly in the database. So, when the Clob object is same in the updateClob method as the Clob object obtained using getClob, the updateRow method of the ResultSet interface does nothing with the Clob object.

When the Clob objects differ, the Clob object in the updateClob method behaves as if the setClob method was issued. See Inserting a Clob Object with the setClob Method.

Replacing Clob Objects

You can replace Clob objects in the following ways:

- Use the EMPTY_CLOB() function to replace the Clob object with the empty Clob object, then insert new data as described under Inserting CLOB Columns by Using the Clob Interface.
- Use the PreparedStatement.setAsciiStream() or setCharacterStream() method to replace the existing Clob object with new CLOB data.
- Use the setClob or updateClob method to replace the existing CLOB objects as explained under Inserting a Clob Object with the setClob Method and Updating Clob Objects with the updateClob Method.

Deleting CLOB Data

To delete CLOB data, the JDBC application uses the SQL DELETE statement to delete the row in the base table.

When the row containing the CLOB column is deleted by the JDBC application, the corresponding CLOB data is automatically deleted by the delete trigger associated with the base table. For information about triggers, see Using SQL/MX Triggers to Delete LOB Data.

See also "NULL and Empty BLOB or Empty CLOB Value" (page 66).

Storing BLOB Data

- "Inserting a BLOB Column by Using the Blob Interface" (page 63)
- "Writing Binary Data to a BLOB Column" (page 64)
- "Inserting a BLOB Column by Using the PreparedStatement Interface" (page 64)
- "Inserting a Blob Object by Using the setBlob Method" (page 64)

Perform operations on BLOB columns that are similar to those operations used on CLOB columns.

- Use the EMPTY_BLOB() function in the insert statement to create an empty BLOB column in the database.
- Use setBinaryStream method of the Blob interface to obtain the InputStream to read BLOB data.
- Use getBinaryStream method of the Blob interface to obtain the OutputStream to write BLOB data.
- Use setBinaryStream of the PreparedStatement interface to write the data to the BLOB column.

Inserting a BLOB Column by Using the Blob Interface

When you insert a row containing a BLOB data type, you can insert the row using an empty BLOB value before the column can be updated with real BLOB data. To insert an empty BLOB value in an SQL/MX database, specify EMPTY_BLOB() function for the BLOB column in the insert statement.

The Type 4 driver scans the SQL string for the EMPTY_BLOB() function and substitutes the next-available data locator.

NOTE:

- The EMPTY_BLOB() function is an HPE NonStop specific function and might not work on other databases.
- Do not use the EMPTY_BLOB() function when using the PreparedStatement interface.

Then, obtain the handle to the empty BLOB column by selecting the BLOB column for update. The following code illustrates how to obtain the handle to an empty BLOB column:

```
Blob myBlob = null;
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("Select myBlobColumn
  from myTable where …For update");
if (rs.next())
  myBlob = rs.getBlob(1);
```

You can now write data to the BLOB column. See Writing Binary Data to a BLOB Column.

NOTE: Limitation: Do not rename the BLOB column in the select query.

Writing Binary Data to a BLOB Column

To write data to the BLOB column, use the Blob interface. The following code illustrates using the setBinaryStream method of the Blob interface to write BLOB data:

```
// Stream begins at position: 1
long pos = 1;
// String containing the binary data
String s ;
// Obtain the output stream to write Blob data
OutputStream os = myBlob.setBinaryStream(pos);
// write Blob data using OutputStream
byte[] myBlobData = s.getBytes();
os.write(myBlobData);
```

The Type 4 driver splits the output stream into chunks and stores the chunks in the LOB table.

Inserting a BLOB Column by Using the PreparedStatement Interface

To insert a BLOB column that has binary data from an InputStream, use the PreparedStatement interface:

```
InputStream inputBinary;
PreparedStatement ps = conn.prepareStatement("insert
    into myTable (myBlobColumn) values (?)");
ps.setBinaryStream(1, inputBinary, length_of_data);
ps.executeUpdate();
```

The Type 4 driver reads the data from InputStream and writes the data to the LOB table. The Type 4 driver substitutes the next-available data locator for the parameter of the BLOB column in the table.

Inserting a Blob Object by Using the setBlob Method

Your JDBC application cannot directly instantiate a Blob object. To perform an equivalent operation:

- Obtain a Blob object by using the getBlob method of the ResultSet interface.
- Insert the Blob object into another row by using the setBlob method of the PreparedStatement interface.

In this situation, the Type 4 driver generates a new data locator and copies the contents of the source Blob into the new Blob object when the application issues the setBlob method of the PreparedStatement interface.

Reading Binary Data from a BLOB Column

To read binary data from the BLOB column, use the Blob interface or InputStream.

Using the Blob interface:

```
// Obtain the Blob from ResultSet
Blob myBlob = rs.getBlob("myBlobColumn");
// Obtain the input stream to read Blob data
InputStream is = myBlob.getBinaryStream();
// read Blob data using the InputStream
byte[] myBlobData;
myBlobData = new byte[length];
is.read(myBlobData, offset, length);
```

Using `InputStream`:

```
// obtain the InputStream from ResultSet
InputStream is = rs.getBinaryStream("myBlobColumn");
// read Blob data using the InputStream
byte[] myBlobData;
myBlobData = new byte[length];
is.read(myBlobData, offset, length);
```

Updating BLOB Data

To update BLOB data, use the methods in the Blob interface or use the `updateBlob` method of the `ResultSet` interface. The Type 4 driver makes changes to the BLOB data directly. Therefore, the Type 4 driver returns false to the `locatorsUpdateCopy` method of the `DatabaseMetaData` interface. Applications do not need to issue a separate update statement to update the BLOB data.

Update BLOB data in the following ways.

- “Updating `Blob` Objects by Using the `updateBlob` Method” (page 65)
- “Replacing `Blob` Objects” (page 65)

Updating `Blob` Objects by Using the `updateBlob` Method

Unlike some LOB support implementations, the Type 4 driver updates the BLOB data directly in the database. So, when the `Blob` object is same in the `updateBlob` method as the object obtained using `getBlob`, the `updateRow` method of the `ResultSet` interface does nothing with the `Blob` object.

When the `Blob` objects differ, the `Blob` object in the `updateBlob` method behaves as if the `setBlob` method was issued. See Inserting a `Blob` Object with the `setBlob` Method.

Replacing `Blob` Objects

You can replace Blob objects in the following ways:

- Use the `EMPTY_BLOB()` function to replace the `Blob` object with the empty `Blob` object.
- Replace an existing `Blob` object in a row by inserting the `Blob` with new data as described under Inserting a BLOB Column Using the Blob Interface.
- Use the `setBinaryStream()` method of the `PreparedStatement` interface to replace the existing `Blob` object with new BLOB data.
- Use the `setBlob` or `updateBlob` methods to replace the existing BLOB objects as explained under Inserting a Blob Object with the setBlob Method and Updating Blob Objects with the UpdateBlob Method.

Deleting BLOB Data

To delete BLOB data, the JDBC application uses the SQL DELETE statement to delete the row in the base table.

When the row containing the BLOB column is deleted by the application, the corresponding BLOB data is automatically deleted by the delete trigger associated with the base table. For information about triggers, see Using SQL/MX Triggers to Delete LOB Data.

See also "NULL and Empty BLOB or Empty CLOB Value" (page 66).

NULL and Empty BLOB or Empty CLOB Value

The data locator can have a NULL value if the BLOB or CLOB column is omitted in the insert statement. The Type 4 driver returns NULL when the application retrieves the value for such a column.

When the application uses the EMPTY_BLOB() function or the EMPTY_CLOB() function to insert empty BLOB or CLOB data into the BLOB or CLOB column, the Type 4 driver returns the Blob or Clob object with no data.

Transactions Involving Blob and Clob Access

You must ensure that your JDBC applications control the transactions when the BLOB columns or CLOB columns are accessed or modified. To control the transaction, set the connection to autocommit OFF mode.

All LOB data access or modification is done under the application's transaction. When the autocommit mode is ON and LOB data is accessed or modified, the Type 4 driver throws the SQLException, `Autocommit is on and LOB objects are involved`.

Access Considerations for Clob and Blob Objects

The Type 4 driver allows all the valid operations on the current Clob object or Blob object, called a LOB object. LOB objects are current as long as the row that contains these LOB objects is the current row. The Type 4 driver throws an SQLException, issuing the following message, when the application attempts to perform operations on a LOB object that is not current:

```
Lob object {object-id} is not current
```

Only one InputStream or Reader and one OutputStream or Writer can be associated with the current LOB object.

- When the application obtains the InputStream or Reader from the LOB object, the Type 4 driver closes the InputStream or Reader that is already associated with the LOB object.
- Similarly, when the application obtains the OutputStream or Writer from the LOB object, the Type 4 driver closes the OutputStream or Writer that is already associated with the LOB object.

6 Managing the SQL/MX Tables for BLOB and CLOB Data

This chapter describes the creation and management of the tables required to support LOB data for database administrators.

Before You Begin Managing LOB Data

BLOB and CLOB are not native data types in an SQL/MX database. But, database administrators can create SQL/MX tables that have BLOB and CLOB columns by using the Type 4 driver or special SQL syntax in MXCI as described in this chapter. For management purposes, CLOB and BLOB data is referred to as large object (LOB) data, which can represent either data type.

Before reading this chapter, also read Architecture for LOB Support, which describes the files for the tables that contain LOB data.

Creating Base Tables that Have LOB Columns

You can write JDBC programs to create base tables that have LOB columns or you can use the SQL/MX conversational interface MXCI as described in this chapter.

Data Types for LOB Columns

The data types for the LOB columns are:

CLOB
: Character large object data

BLOB
: Binary large object data

NOTE: The CLOB and BLOB data type specification is special syntax that is allowed for use in base tables accessed by the Type 4 driver described in this manual.

Using MXCI To Create Base Tables that Have LOB Columns

Before using the procedure to create the tables, note that when using MXCI to create base tables, you must enter the following special command in the MXCI session to enable the base-table creation of tables that have LOB (BLOB or CLOB) columns:

```
CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE'
```

Follow these steps to create a base table that has LOB columns:

1. At the OSS prompt, enter the following to invoke the SQL/MX utility MXCI:

   ```
   mxci
   ```

2. Type the following command to enable creating tables that have LOB columns:

   ```
   CONTROL QUERY DEFAULT JDBC_PROCESS 'TRUE';
   ```

3. Type the CREATE TABLE statement; for example, you might use the following simple form of the statement:

   ```
   CREATE TABLE table1 (c1 INTEGER NOT NULL, c2 CLOB, c3 BLOB, PRIMARY
   KEY(c1));
   ```

where

`table1`
: The name of the base table.

NOTE: If different LOB tables are used for storing BLOB or CLOB data, the base table name for a table with BLOB or CLOB columns must be unique across all catalogs and schemas. Otherwise, the driver will give incorrect data to the application in cases where the LOB tables used get erroneously switched or changed.

`c1`
: Column 1, defined as the INTEGER data type with the NOT NULL constraint.

`c2`
: Column 2, defined as the CLOB data type.

`c3`
: Column 3, defined as the BLOB data type.

`PRIMARY KEY`
: Specifies c1 as the primary key.

Use this example as the archetype for creating base tables. For information about valid names for tables (table1) and columns (c1, c2, and c3) and for information about the CREATE TABLE statement, see the *HPE NonStop SQL/MX Release 3.2.1 Reference Manual*.

Using JDBC Programs To Create Base Tables that Have LOB Columns

When using a JDBC Program to create base tables that have LOB columns, put the CREATE TABLE statements in the program as you would any other SQL statement. For an example of the CREATE TABLE statement, see the discussion Using MXCI to Create Base Tables that Have LOB Columns.

Managing LOB Data by Using the Lob Admin Utility

Use the Lob Admin Utility (T4LobAdmin) for the following tasks:

- Creating the LOB table (a table that holds LOB data).
- Creating the SQL/MX triggers for the LOB columns in the base tables to ensure that orphan LOB data does not occur in a LOB table.

NOTE: If you are creating triggers, ensure that the base table that contains the `CLOB` column or `BLOB` column has already been created.

Information about using the Lob Admin Utility is described under these topics.

- "Running the Lob Admin Utility" (page 69)
- "Help Listing from the Type 4 Lob Admin Utility" (page 70)
- "Creating LOB Tables" (page 70)

Running the Lob Admin Utility

Run the T4LobAdmin utility in the OSS environment.

The format of the command is:

```
java [java_options] com.tandem.t4jdbc.T4LobAdmin [prog_options] [table_name]
```

java_options

The `java_options` should specify the Type 4 driver properties in a properties file on the `java` command line in the `-D` option.

`-Dt4sqlmx.properties=`*`properties-file-name`*

where the properties file should include the following Type 4 driver properties, as applicable:

`blobTableName`

Specifies LOB table for `BLOB` columns. Required if `BLOB` columns are involved. See "LOB Table Name Properties" (page 45).

`clobTableName`

Specifies the LOB table for `CLOB` columns. Required if `CLOB` columns are involved. See "LOB Table Name Properties" (page 45).

`url`

URL for the Type 4 driver connection. See url Property.

`usr`

User name for the Type 4 driver connection. See user Property.

`password`

Password associated with the user. See password Property.

program_options

prog_option	Description
-help	Displays help information
-exec	Runs the SQL statements that are generated.
-create	Generates SQL statements to create LOB tables. These statements describe the architecture of the tables and, therefore, provide a description of the LOB tables.
-trigger	Generates SQL statements to create triggers for the designated table. The base table must exist.

prog_option	Description
-drop	Generate SQL statements to drop triggers for the designated table. The table must exist.
-out	Writes the SQL statements to a specified file.
-unicode -	Generates SQL statements to create unicode LOB tables <CLOB only>.
-bigblock	Generates SQL statements to create LOB column size of 24K bytes and attribute block size of 32K.

table_name

The table_name represents a base table that contains LOB columns. The table_name is of the form:

```
[catalogName.][schemaName.]baseTableName
```

For information about catalog, schema, and table names, see the *HPE NonStop SQL/MX Release 3.2.1 Reference Manual*.

Help Listing from the Type 4 Lob Admin Utility

To display help for the Type 4 Lob Admin Utility, type:

```
java -Dt4sqlmx.properties=t4lob.prop com.tandem.t4jdbc.T4LobAdmin -help
```

Example 4 Example 1. Type 4 Lob Admin Utility Help

```
Hewlett-Packard T4 Lob Admin Utility 1.0
(c) Copyright 2004 Hewlett-Packard Development Company, LP.

java [<java_options>] T4LobAdmin [<prog_options>] [<table_name>]

<java_options> is:
  [-Dt4sqlmx.properties=<properties file>]
where <properties file> has values for the following:
  clobTableName - CLOB table name
  blobTableName - BLOB table name
  url - URL used for the Type 4 connection
  user - User name for the Type 4 connection
  password - Password for associated with the user

<prog_options> is:
  [-exec] [-create] [-trigger][-unicode] [-spj] [-help] [-drop] [-out <filename>] [-bigblock]
where -help - Display this information.
  -exec - Execute the SQL.
  -create - Generate SQL statements to create LOB tables.
  -trigger - Generate SQL statements to create triggers for <table_name>.
  -unicode - Generate SQL statements to create unicode LOB tables <CLOB only>.
  -drop - Generate SQL statements to drop triggers for <table_name>.
  -out - Write the SQL statements to <filename>.
  -spj - Generate SQL statements to create and execute tables for SPJ LOB usage.
  -bigblock- Generates SQL statements to create LOB column size of 24K bytes and
     attribute block size of 32K.

<clobTableName> | <blobTableName> is:
  <catalogName>.<schemaName>.<lobTableName>

<table_name> is:
  [<catalogName>.][<schemaName>.]<baseTableName>

<baseTableName> is the table that contains LOB column(s). TableName> is the table that contains the LOB data.
```

Creating LOB Tables

Except as noted in the following, use the `-create` and `-execute` options of the Lob Admin Utility to create LOB tables.

NOTE: Partitioned LOB tables must be manually created. You cannot use the Lob Admin Utility if your site needs partitioned LOB tables. Do not use the `-execute` option of the Lob Admin Utility. Follow these steps to manually create partitioned LOB tables:

1. Use the `-create` and `-out` options of the Lob Admin Utility to have SQL statements written to a file.
2. Modify the generated SQL statements as needed for your partitioning requirements.
3. Add the modified SQL statements to an MXCI script file.
4. Move the MXCI script file to the OSS environment, and run it.

Using SQL/MX Triggers to Delete LOB Data

Use the Type 4 Lob Admin Utility to generate triggers that delete LOB data from the LOB table when the base row is deleted. These triggers ensure that orphan LOB data does not occur in the LOB table. To manage the triggers, use these Type 4 Lob Admin Utility options:

`-trigger`
: Generates SQL statements to create triggers.

`-drop`
: Generates SQL statements to drop triggers.

`-exec`
: Executes the SQL statements that are generated.

For example, the following command generates the SQL statements to create the triggers for the base table `sales.paris.pictures`, which contains a `BLOB` column, and executes those statements. Note: This command must be typed on one line.

```
java -Dt4sqlmx.blobTableName=sales.paris.lobTable4pictures
  com.tandem.t4jdbc.T4LobAdmin
  -trigger
  -exec sales.paris.pictures
```

Backing Up and Restoring LOB Columns

For basic information about backing up and restoring databases, see the *HPE NonStop SQL/MX Release 3.2.1 Management Guide*. That discussion describes the special considerations you must take for backing up and restoring tables that have LOB columns implemented for the Type 4 driver (because both base tables and LOB tables are involved).

- When backing up and restoring a base table, make sure that the name of the table is unchanged. LOB data is not be accessible after restoration if the base table name has changed.
- Triggers cannot be restored by using Backup and Restore 2.0. You can capture the DDL for the CREATE statements executed for these objects and use this information to manually recreate these objects after a Restore operation.
- Make sure that the time interval in the backup of the base tables and LOB tables is not large. A smaller time interval ensures that the data referred to by the base table is present in the LOB table.

Limitations of LOB Data (CLOB and BLOB Data Types)

Limitations of the CLOB and BLOB data types, collectively referred to as LOB data, are:

- LOB columns can only be in the target column list of these SQL statements:
  - INSERT statement
  - Select list of a SELECT statement
  - Column name in the SET clause of an UPDATE statement
- LOB columns cannot be referenced in the SQL/MX functions and expressions.
- LOB data is not deleted from the LOB table when the base row is deleted unless a trigger is established. For information about triggers, see Using SQL/MX Triggers to Delete LOB Data.
- LOB data is not accessible if the base table name is changed.
- LOB columns cannot be copied to another table by using the SQL/MX utility commands.
- The name of a base table that has CLOB or BLOB columns must be unique across all catalogs and schemas when more than one of these base tables share a single LOB table.

NOTE: Adding a trigger can affect up to three schemas. For each schema, you must either own the schema or be the super ID.

- The schema where the trigger is created.
- The schema where the subject table (LOB table) exists.
- The schema where the referenced table (base table) exists.

7 Module File Caching (MFC)

MFC Overview

Applications that use the JDBC Type 4 driver might encounter repeated SQL compiles while using statements and `PreparedStatements`. To overcome this issue, a driver side Most Frequently Used (MFU) cache is used. However, the current design of the driver side cache consumes large amount of memory. The size of the MXOSRVR process memory is directly proportional to the size of the driver side cache configured through the `maxStatements` property. Higher `maxStatements` value (driver side cache size) can breach the MXOSRVR process memory limit of 2 GB on the NonStop platform.

The Module File Caching (MFC) feature shares the statement and `PreparedStatement` plans among the NonStop SQL/MX database connections. The MFC helps in reducing the SQL compilation time during JDBC or ODBC application's steady state, thereby reducing resource consumption.

NOTE: Module File Caching is supported only on systems running J06.07 and later J-series RVUs and H06.18 and later H-series RVUs. Starting with SQL/MX Release 3.2, MFC supports the `BigNum` data type in parameterized queries.

MFC Design

For information on the MFC design, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

Configuring MFC

For information on configuring MFC, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

Enabling MFC

MFC for the JDBC Type 4 driver is enabled through the MXCS configuration, and SQL modules are created automatically when statements are prepared. A client program can explicitly turn OFF MFC if modules are not required.

The following property is required to use MFC in an application that uses the JDBC Type 4 driver:

`enableMFC`

To enable MFC:

- Change the data source configuration settings on the server side. For more information on changing the data source settings, see the *HPE NonStop Connectivity Service Manual*.
- The property `enableMFC` must be set to ON. The default value is ON.

To disable MFC, the value of this property must be set to OFF. For example:
`-Dt4sqlmx.enableMFC = OFF`.

NOTE: There is no need to change any data source configuration settings to disable MFC.

For more information on MFC, see:

- *Module File Cache for NonStop SQL/MX* at http://www.hpe.com/support/Module_File_Cache_NonStop_SQLMX_White_Paper
- *Distributed transactions (XA) on HPE NonStop systems* at http://www.hpe.com/info/nonstop-docs

MFC Usage Scenarios

The benefits of lower processor utilization and the reduction in subsequent compile times mentioned in this section are applicable on the NonStop Server.

- JDBC T4 applications using `java.sql.Connection.Statement()` and `java.sql.Connection.prepareStatement()` have the benefit of lower processor utilization and better response time.
- JDBC T4 applications using complex queries through prepare calls, have higher benefit of lower processor utilization, lower memory consumption, and better response time. For example, Hibernate generated queries.
- JDBC T4 applications with higher driver-side cache size can benefit from lower memory utilization by using the combination of MFC and a lower driver-side MFU cache.

  For example, an application with `t4sqlmx.maxStatements=1000` can benefit by changing to `t4sqlmx.maxStatements=600` and MFC.

  In this example, all statements are cached in MFC and the most frequently used statements are cached in the driver side MFU cache.
- JDBC T4 applications, which have large number of statements that are SQL compiled at the startup, can benefit from reduced (re-) startup time by using the MFC feature. In situations, where servers are configured for automatic restart, the startup time is reduced.

  JDBC T4 applications experiencing high memory swapping can benefit from reducing the driver side cache and enabling MFC. Reduce the driver side cache and enable MFC in the scenarios, where the MXOSRVR process exceeds 1 GB in size frequently.
- For JDBC T4 applications where the large number of MXOSRVR processes is configured per processor, enable MFC to reduce swapping. The other option is to increase the physical memory per processor to reduce swapping.
- MFC combined with limited driver side cache is recommended for JDBC T4 applications, where the number of distinct queries is not known or not fixed.

MFC Tuning Recommendations

When the number of connections required to be configured per processor is more (for example, more than 20 connections per processor), use MFC for less memory utilization.

If an application has a small number of OLTP queries where there is no memory pressure and memory pressure is heavy in `execute()` and `fetch()`, the MFC performance result will be close to that of the T4 driver cache with MFC consuming lesser memory. In such applications, when a small number of connections are configured per processor, configure only the JDBC T4 driver cache because there is no major difference between MFC and the T4 driver cache.

For applications with small number of queries, Driver Side Cache (DSC) provides better response time. For applications that have a large number of queries, use a combination of DSC and MFC, which results in better memory usage and similar response time of DSC. For typical applications, find the number of most frequently used statements and configure them as DSC and the rest as

MFC. If the most frequently used statements number is not known, it is recommended that you configure a number closer to 60% of total queries in the application.

For example, if the application has 1000 unique queries, configure DSC (`t4sqlmx.maxStatements`) to 600. The applications with more number of queries configured in the DSC, cause memory swap. Therefore, reduce the DSC number and increase the queries in the MFC.

The JDBC/T4 driver, in memory, caches the most frequently used statements.

MFC Limitations

MFC is not a replacement for the JDBC T4 Driver Side Cache (DSC); the JDBC T4 driver side cache has a better response time than MFC.

When the module is created the first time, the MFC-related module file creation process consumes more time for compilation.

For more information on MFC limitations, see the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1*.

8 Type 4 Driver Compliance

Compliance Overview

The Type 4 driver conforms where applicable to the Oracle JDBC 3.0 API specification. However, this driver differs from the JDBC standard in some ways. This subsection describes the JDBC methods that are not supported, the methods and features that deviate from the specification, and features that are Hewlett Packard Enterprise extensions to the JDBC standard. JDBC features that conform to the specification are not described in this subsection.

In addition, this chapter lists features of NonStop SQL/MX and SQL/MP that are not supported by the NonStop JDBC type 4 Driver, other unsupported features, and restrictions.

Unsupported Features

The following methods in the `java.sql` package throw an `SQLException` with the message "Unsupported feature - method-name":

Method	Comments

`CallableStatement.getArray(int parameterIndex)`
`CallableStatement.getArray(StringparameterName)`
`CallableStatement.getBlob(int parameterIndex)`
`CallableStatement.getBlob(StringparameterName)`
`CallableStatement.getClob(int parameterIndex)`
`CallableStatement.getClob(StringparameterName)`
`CallableStatement.getObject(int parameterIndex, Map map)`
`CallableStatement.getObject(String parameterName, Map map)`
`CallableStatement.getRef(int parameterIndex)`
`CallableStatement.getRef(StringparameterName)`
`CallableStatement.getURL(int parameterIndex)`
`CallableStatement.getURL(String parameterName)`
`CallableStatement.executeBatch()`
	The particular `CallableStatement` method is not supported.

`Connection.releaseSavepoint(Savepoint savepoint)`
`Connection.rollback(Savepoint savepoint)`
`Connection.setSavepoint()`
`Connection.setSavepoint(String name)`
	The particular `Connection` methods are not supported.

Method	Comments

`PreparedStatement.setArray(int parameterIndex, Array x)`
`PreparedStatement.setRef(int parameterIndex, Ref x)`
`PreparedStatement.setURL(int parameterIndex, URL x)`
	The particular `PreparedStatement` method is not supported.

`ResultSet.getArray(int columnIndex)`
`ResultSet.getArray(String columnName)`
`ResultSet.getObject(int columnIndex, Map map)`
`ResultSet.getObject(String columnName, Map map)`
`ResultSet.getRef(int columnIndex)`
`ResultSet.getRef(String columnName)`
`ResultSet.getURL(int columnIndex)`
`ResultSet.getURL(String columnName)`
`ResultSet.updateArray(int columnIndex)`
`ResultSet.updateArray(String columnName)`
`ResultSet.updateRef(int columnIndex)`
`ResultSet.updateRef(String columnName)`
	The particular `ResultSet` methods are not supported.

When used for access to SQL/MP user tables, the following additional methods in the `java.sql` package throw an `SQLException` with the message "Unsupported feature - method-name":

Method	Comments
`PreparedStatement.setBlob(intparameterIndex, Blob x)` `PreparedStatement.setClob(intparameterIndex, Clob x)`	The particular `PreparedStatement` methods are not supported for access of SQL/MP user tables only.

`ResultSet.getBlob(int columnIndex)`
`ResultSet.getBlob(String columnName)`
`ResultSet.getClob(int columnIndex)`
`ResultSet.getClob(String columnName)`
`ResultSet.updateBlob(int columnIndex)`
`ResultSet.updateBlob(String columnName)`
`ResultSet.updateClob(int columnIndex)`
`ResultSet.updateClob(String columnName)`
	The particular `ResultSet` methods are not supported for access of SQL/MP user tables only.

The following methods in the `java.sql` package throw an `SQLException` with the message "Auto generated keys not supported":

Method	Comments

`Connection.prepareStatement(String sql, int autoGeneratedKeys)`
`Connection.prepareStatement(String sql, int[] columnIndexes)`
`Connection.prepareStatement(String sql, String[] columnNames)`
	Automatically generated keys are not supported.

`Statement.executeUpdate(String sql, int autoGeneratedKeys)`
`Statement.executeUpdate(String sql, int[] columnIndexes)`
`Statement.executeUpdate(String sql, String[] columnNames)`
`Statement.getGeneratedKeys()`
	Automatically generated keys are not supported.

The following methods in the `java.sql` package throw an `SQLException` with the message "Data type not supported":

Method	Comments
`CallableStatement.getBytes(int parameterIndex)` `CallableStatement.setBytes(StringparameterIndex, bytes[] x)`	The particular data type is not supported.

The following interfaces in the `java.sql` package are not implemented in the Type 4 driver:

Interrface	Comments

`java.sql.Array`
`java.sql.Ref`
`java.sql.Savepoint`
`java.sql.SQLData`
`java.sql.SQLInput`
`java.sql.SQLOutput`
`java.sql.Struct`
	The underlying data types are not supported by SQL/MX.

The following interfaces in the `javax.sql` package are not implemented in the Type 4 driver:

Method	Comments

`javax.sql.RowSet`
`javax.sql.RowSetInternal`
`javax.sql.RowSetListener`
`javax.sql.RowSetMetaData`
`javax.sql.RowSetReader`
`javax.sql.RowSetWriter`
	`RowSet` is not implemented in the JDBC Driver for SQL/MX. You can, however, download reference implementation of `RowSet` from the Oracle Web site (http://java.sun.com/developer/earlyAccess/jdbc/jdbc-rowset.html).

For additional information about deviations for some methods, see Deviations.

Deviations

The following table lists methods that differ in execution from the JDBC specification. When an argument in a method is ignored, the Type 4 driver does not throw an `SQLException`, thus allowing the application to continue processing. The application might not obtain the expected results, however. Other methods listed do not necessarily throw an `SQLException`, unless otherwise stated, although they differ from the specification.

NOTE: `java.sql.DatabaseMetaData.getVersionColumns()` method mimics the `java.sql.DatabaseMetaData.getBestRowIdentifier()` method because SQL/MX does not support SQL_ROWVER (a columns function that returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction).

Method	Comments
`java.sql.DatabaseMetaData.getColumns(String catalog, String schemaPattern,String tableNamePattern, String columnNamePattern)`	The column is added to the column data, but its value is set to NULL because SQL/MXdoes not support the column type for types as follows: SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_TABLE, and

Method	Comments
	SOURCE_DATA_TYPE.
`java.sql.DatabaseMetaData.getTables(String catalog, String schemaPattern, String[] types)`	The column is added to the column data, but its value is set to NULL because SQL/MXdoes not support the column type for types as follows: TYPE_CAT, TYPE_SCHEMA, TYPE_NAME, SELF_REFERENCING_COL_NAME, and REF_GENERATION.
`java.sql.DatabaseMetaData.getUDTs(String catalog, String schemaPattern, String tableNamePattern, int[] types)`	BASE_TYPE is added to the column data, but its value is set to NULL because SQL/MXdoes not support the base type.
`java.sql.DatabaseMetaData.getVersionColumns()`	Mimics the `DatabaseMetaData.getBestRowIdentifier()` method because SQL/MX does not support SQL_ROWVER (a columns function that returns the column or columns in the specified table, if any, that are automatically updated by the data source when any value in the row is updated by any transaction).
`java.sql.Connection.createStatement(...)java.sql.Connection.prepareStatement(...)`	The Type 4 driver does not support the scroll-sensitive result set type, so an `SQLWarning` is issued if an application requests that type. The result set is changed to a scroll-insensitive type.
`java.sql.Connection.setReadOnly(...)`	The read-only attribute is ignored.
`java.sql.ResultSet.setFetchDirection(...)`	The fetch direction attribute is ignored.
`java.sql.Statement.setEscapeProcessing(...)`	Because SQL/MX parses the escape syntax, disabling escape processing has no effect.
`java.sql.Statement.setFetchDirection(...)`	The fetch direction attribute is ignored.
`java.sql.Statement.setQueryTimeout(int seconds)`	This is based on the JDBC Driver property `closeConnectionUponQueryTimeout` value. The following is the behavior of the JDBC Driver for the values IGNORE/DEFAULT for the property `closeConnectionUponQueryTimeout`: IGNORE = Any value (> 0) set by calling `Statement.setQueryTimeout()` has no effect. The Statement continues to block the current thread until the statement is run. DEFAULT = This is the value set if the property is not specified. Any value (> 0) set by calling the `Statement.setQueryTimeout()` causes an SQLException to be raised when this is the value.

Hewlett Packard Enterprise Extensions

The following Hewlett Packard Enterprise extensions to the JDBC standard are implemented in the Type 4 driver.

Internationalization of Messages

The Type 4 driver is designed so that Java messages can be adopted for various languages. The error messages are stored outside the source code in a separate property file and retrieved dynamically based on the locale setting. The error messages in different languages are stored in separate property files based on the language and country. This extension does not apply to all messages that can occur when running JDBC applications.

For details, see Localizing Error and Status Messages.

Conformance of DatabaseMetaData Methods' Handling of Null Parameters

This topic describes how the Type 4 driver determines the value of null parameters passed as a parameter value on `DatabaseMetaData` methods. Since other vendors might implement the JDBC specification differently, this information explains the Type 4 driver results on the affected queries.

This implementation applies to methods that take parameters that can represent a pattern. The names of these parameters have the format:

`attributePattern`

The many methods of the `java.sql.DatabaseMetaData` class are affected; for example, the `getColumns()` method.

For another example, schema is the attribute in the parameter `schemaPattern`, which is a parameter to the `java.sql.ResultSet.getAttributes` method.

```
public ResultSet getAttributes(String catalog,
     String schemaPattern,
     String typeNamePattern,
     String attributeNamePattern)
     throws SQLException
```

If the application passes a null value, the null is treated as follows:

- If a parameter name contains the suffix `Pattern`, the null is interpreted as a % wildcard.
- If the parameter name does not contain the suffix `Pattern`, nulls are interpreted as the default value for that parameter.

Using this example, null parameters are interpreted as follows:

catalog
: the default catalog name

schemaPattern
: a % wildcard and retrieves data for all schemas of the specified catalog

Type 4 Driver Conformance to SQL Data Types

JDBC Data Types

The following table shows the JDBC data types that are supported by Type 4 driver and their corresponding SQL/MX data types:

JDBC Data Type	Support by JDBC Type 4 Driver for NonStop SQL/MX	SQL/MX Data Type
`Types.Array`	No	N.A.
`Types.BIGINT`	Yes	LARGEINT
`Types.BINARY`	Data type is mapped by SQL/MX . Data type varies from that used for table creation.	CHAR(n)[1]
`Types.BIT`	Data type is mapped by SQL/MX . Data type varies from that used for table creation.	CHAR(1)
`Types.BLOB`	Yes	LARGEINT
`Types.CHAR`	Yes	CHAR(n)
`Types.CLOB`	Yes	LARGEINT
`Types.DATE`	Yes	DATE
`Types.DECIMAL`	Yes	DECIMAL(p,s)
`Types.DISTINCT`	No	N.A.
`Types.DOUBLE`	Yes	DOUBLE PRECISION
`Types.FLOAT`	Yes	FLOAT(p)
`Types.INTEGER`	Yes	INTEGER
`Types.JAVA_OBJECT`	No	N.A.
`Types.LONGVARBINARY`	Data type is mapped by SQL/MX . Data type varies from that used for table creation.	VARCHAR(n)[1]
`Types.LONGVARCHAR`	Yes Maximum length is 4018	VARCHAR[(n)]
`Types.NULL`	No	N.A.
`Types.NUMERIC`	Yes	NUMERIC(p,s)
`Types.REAL`	Yes	FLOAT(p)
`Types.REF`	No	N.A.
`Types.SMALLINT`	Yes	SMALLINT
`Types.STRUCT`	No	N.A.
`Types.TIME`	Yes	TIME
`Types.TIMESTAMP`	Yes	TIMESTAMP
`Types.TINYINT`	Data type is mapped by SQL/MX . Data type varies from that used for table creation.	SMALLINT
`Types.VARBINARY`	Data type is mapped by SQL/MX . Data type varies from that used for table creation.	VARCHAR(n)[1]

JDBC Data Type	Support by JDBC Type 4 Driver for NonStop SQL/MX	SQL/MX Data Type
`Types.VARCHAR`	Yes	VARCHAR(n)
`Types.BOOLEAN`	Data type is mapped by SQL/MX . Data type varies from that used for table creation.	CHAR(1)
`Types.DATALINK`	No	N.A.

[1] Because of mapping provided by SQL/MX, a ResultSet.getObject() method returns a string object instead of an array of bytes.

The Type 4 driver maps the following data types to the JDBC data type `Types.OTHER`:

SQL/MX Data Type	SQL/MP Data Type
INTERVAL YEAR(p)	DATETIME YEAR
INTERVAL YEAR(p) TO MONTH	DATETIME YEAR TO MONTH
INTERVAL MONTH(p)	DATETIME YEAR TO HOUR
INTERVAL DAY(p)	DATETIME YEAR TO MINUTE
INTERVAL DAY(p) TO HOUR	DATETIME MONTH
INTERVAL DAY(p) TO MINUTE	DATETIME MONTH TO DAY
INTERVAL DAY(p) TO SECOND	DATETIME MONTH TO HOUR
INTERVAL HOUR(p)	DATETIME MONTH TO SECOND
INTERVAL HOUR(p) TO MINUTE	DATETIME DAY
INTERVAL HOUR(p) TO SECOND	DATETIME DAY TO HOUR
INTERVAL MINUTE(p)	DATETIME DAY TO MINUTE
INTERVAL MINUTE(p) TO SECOND	DATETIME DAY TO SECOND
INTERVAL SECOND(p)	DATETIME HOUR
	DATETIME HOUR TO MINUTE
	DATETIME MINUTE
	DATETIME MINUTE TO SECOND
	DATETIME SECOND
	DATETIME FRACTION

NOTE: For more information on conversion performed between Java Object Types and Target JDBC Types, see JDBC 3.0 specification: http://java.cnam.fr/iagl/biblio/spec/jdbc-3_0-fr-spec.pdf.

Floating-Point Support

The Type 4 driver supports only IEEE floating-point data to be passed between the application client and the Type 4 driver.

NOTE: Any data access to or from a column with Tandem floating-point format in SQL/MP, might result in loss of precision of the data.

JDBC Type 4 Driver Features

The JDBC Type 4 driver supports the following features:

- Row size limit is increased to align with maximum block size 32768 for MX tables.
- A clustering key length up to 2048 bytes is allowed for range and hash partitioned MX tables.
- The precision of the NUMERIC data type is extended up to 128 digits for signed and unsigned values.

Unsupported NonStop SQL Features

- "Unsupported SQL/MX Features" (page 83)
- "Unsupported SQL/MP Features" (page 83)

Unsupported SQL/MX Features

- MXCI commands
- SQL/MX utility commands
- SQL Statements
  - Transaction Control statements
  - Resource control and optimization statements
  - Object naming statements
  - Alias statements

Refer to the NonStop SQL/MX documentation set for a complete list of supported features.

Unsupported SQL/MP Features

- SQL/MP DDL support
- Columns described with the character sets ISO8859/2 through ISO8859/9
- Utility commands
- Embedded-only SQL/MP statements
- Transaction Control statements
- Stored procedures

Refer to the NonStop SQL/MP documentation set for a complete list of supported features.

Other Unsupported Features

These features are not required for JDBC 3.0 compliance, and they are not supported by the NonStop JDBC Type 4 Driver.

- Multiple result sets returned by batch statements.
- Database savepoint support. (Not provided in SQL/MX)
- Retrieval of auto generated keys.
- Transform group and type mapping.
- Relationship between connector architecture and JDBC 3.0 SPI.
- Secured socket communication or encryption for the interaction between the NonStop JDBC Type 4 Driver and MXCS. However, secure socket communication or encryption for

Windows-based clients are supported. For more information, see “Providing a secure JDBC connection using NonStop SSL” (page 112)

- Security context (user name and password) implicit propagation from AppServer to the NonStop JDBC Type 4 Driver.
- IPV6 protocol stack. (IPV6 addressing is emulated over IPV4 on the MXCS server side)

Restrictions

- The Type 4 driver supports only database features that are supported by NonStop SQL/MX and SPJ. Therefore, the Type 4 driver is not fully compliant with JDBC 3.0 specifications.
- The Type 4 driver depends on MXCS for all server side manageability related features.
- JDBC Type 4 Driver for SQL/MX Release 3.x cannot connect to a NonStop system running a SQL/MX Release 2.x version of MXCS server objects.

java.sql.connection.setAutoCommit(boolean autoCommit())

Sets the auto-commit mode of this connection to the given state (true or false). If a connection is in auto-commit mode, then the SQL statements will be executed and committed as individual transactions. Otherwise, the SQL statements are grouped into transactions that are terminated by a call to either the method commit or method rollback. By default, new connections are in auto-commit mode. If the value of auto-commit is changed in the middle of a transaction, the current transaction is committed. If `setAutoCommit` is called and the value for auto-commit is not changed from its current value, it is treated as a NO-OP.

9 Tracing and Logging Facilities

The Type 4 driver provides two tracing and logging facilities:

MXCS (server-side) tracing (logging) is enabled by configuring MXCS. For information about MXCS tracing, see the NSM/Web online help.

Standard JDBC Tracing and Logging Facility

The JDBC standard provides a logging and tracing facility, which allows tracing JDBC method calls by setting the log writer. To set the log writer, either call the setLogWriter() method on the `DriverManager` class or call the `setLogWriter()` method on the `DataSource` class (or `ConnectionPoolDataSource` class).

- A `DriverManager` log writer is a character output stream to which all logging and tracing messages for all connections made through the `DriverManager` are printed. This stream includes messages printed by the methods of this connection, messages printed by methods of other objects manufactured by the connection, and so on. The `DriverManager` log writer is initially null, that is, the default is for logging to be disabled.

  For information about using the setLogWriter method, see the DriverManager class API (http://download.oracle.com/javase/1.5.0/docs/api/java/sql/DriverManager.html).

- A `DataSource` log writer is a character output stream to which all logging and tracing messages for this data source are printed. This stream includes messages printed by the methods of this object, messages printed by methods of other objects manufactured by this object, and soon. Messages printed to a data-source-specific log writer are not printed to the log writer associated with the `java.sql.DriverManager` class. When a `DataSource` object is created, the log writer is initially null; that is, the default is for logging to be disabled.

  For information about using the setLogWriter() method, see the DataSource interface API:

  http://download.oracle.com/javase/1.5.0/docs/api/javax/sql/DataSource.html

The Type 4 Driver Logging Facility

- “Controlling the QueryExecuteTime Logging Facility” (page 88)
- “QueryExecuteTime logging Message Format” (page 88)
- “Examples of QueryExecuteTime Logging Output” (page 88)

The Type 4 driver Logging facility allows you to retrieve internal tracing information, which you can use in debugging the driver. It also allows you to capture error and warning messages.

In addition to the standard JDBC tracing and logging facility, the Type 4 driver provides an independent logging facility (Type 4 Driver Logging).

The Type 4 Driver Logging provides the same level of logging and tracing as the standard JDBC tracing and logging facility with the following additional information:

- More detail about the internals of the Type 4 driver and internal tracing information.
- Type 4 driver performance-tuning information.
- Finer control over the amount and type of logging information.
- Error and warning messages.

Accessing the Type 4 Driver Logging Facility

The Type 4 Driver Logging facility is based on the `java.util.logging` package. The Type 4 driver instantiates a `java.util.logging.Logger` class and names the logger `com.tandem.t4jdbc.logger`.

Your JDBC program can access the Type 4 driver logger directly by calling the `java.util.logging.Logger` static method `getLogger(String)`.

For example

```
String t4Logger = java.util.logging.Logger.getLogger(“com.tandem.t4jdbc.logger”);
```

Controlling Type 4 Driver Logging Output

The Type 4 driver provides two properties that you can use to control logging output.

- `T4LogLevel`— Specifies the level of logging. For more information, see T4LogLevel Property.
- `T4LogFile`— Specifies the file to which the driver is to write logging information. For more information, see T4LogFileProperty.

If the application sets several property values, see “Precedence of Property Specifications” (page 40) to determine which setting applies.

Message Format

The format of the trace output is

```
sequence-number ~ time-stamp ~ thread-id
~ [connection-id] ~ [server-id] ~ [dialogue-id]
~ [class].[method][(parameters)] ~ [text]
```

`sequence-number`

A unique sequence number in increasing order.

`time-stamp`

The time of the message, for example `10/17/2004 12:48:23`

`thread-id`

The thread identifier within the JVM.

`connection-id`

If applicable, a unique ID for the connection associated with the message.

server-id

If applicable, information about the MXCS server associated with the message. The `server-id` is of the form:

`TCP:`*`node-name.server-name/port-number`*`:NonStopODBC`

where

node-name

The name of the NonStop server node.

server-name

The Guardian name of the server.

port-number

The port to which the server is connected.

For example:

`TCP:\banshee-tcp.$Z0133/46003:NonStopODBC`

dialogue-id

If applicable, the *dialogue-id* used for the MXCS connection.

class

If applicable, the name of the class that issued the logging request.

method

If applicable, the name of the method that issued the logging request.

parameters

An optional set of parameters associated with the method.

text

Optional textual information for the message.

NOTE: The tilde (~) character separates message parts. This separator allows you to format the message using tools, such as Excel, Word, UNIX sort, and so forth. For example, you can format and sort messages based on sequence number or thread ID. You can edit the log file and change the separator (the tilde) to any character you want. When possible, numbers (such as `thread-id` and `sequence-number`) are pre-pended with zeros (0) to allow for readable formatting.

Examples of Logging Output

- Output where `T4LogLevel` is set to SEVERE

```
0000001 ~ 10/22/2004 10:22:19 ~ 001234 ~ 0049934 ~ ~ ~
 ~ SQLException Operation cancelled.
    SQLSTATE = S1008
SQLCODE = 01118
```

- Output where `T4LogLevel` is set to `FINER`

```
0000006 ~ 10/22/2004 10:34:45 ~ 001234 ~ 0049934 ~ FetchRowSetMessage ~ marshal
~ Entering FetchRowSetMessage.marshal( en_US
, 48345
, STMT_MX_8843
, 5
, 4192,
, 0
, 0 )
```

Controlling the QueryExecuteTime Logging Facility

The Type 4 driver provides two properties that you can use to control the QueryExecute logging output:

- queryExecuteTime: Specifies the time limit to log information about statements which are taking more than the specified time for execution. For information on using this property, see “queryExecuteTime” (page 49).
- T4QueryExecuteLogFile: Specifies the file in which the driver must log information. For information on using this property, see “T4QueryExecuteLogFile” (page 52).

If the application sets several property values, see Precedence of Property Specifications to determine which setting applies.

QueryExecuteTime logging Message Format

The format of the trace output is:

```
sequence-number ~ time-stamp ~ ~ ~ ~ statement-id.SQL query
~ TIME TAKEN execution-time ms
```

where,

sequence-number is a unique sequence number in increasing order.

time-stamp is the time at which the message is generated. For example, 10/17/2004 12:48:23.

statement-id is the statement identifier within the JDBC T4 driver.

SQL query that is consuming more than the specified time.

execution-time is the time taken to execute the SQL query.

Examples of QueryExecuteTime Logging Output

The following output is an example of QueryExecuteTime logging:

```
00000394 ~ Jul 7, 2011 10:08:11 AM GMT ~ 10 ~ ~ ~ ~
SQL_CUR_MFC00000001.SELECT COF_NAME, PRICE FROM ctstable2() ~
TIME TAKEN 47 ms
```

10 Migration and Compatibility

JDBC Drivers

Driver You Are Migrating From	Code Changes	Configuration Changes	Considerations
JDBC Type 3 Driver	No	url, catalog, schema, logging facilities	N.A.
JDBC Driver for SQL/MX (JDBC/MX), a Type 2 driver	No	url, catalog, schema, logging facilities; potential property value changes (verify using Summary of the Type 4 Driver Properties)	Application is not portable if using: • HPE extension cpqPrepare • Unicode in `CLOB` data (not supported) • LOB with autocommit ON (not supported)
SequeLink JDBC Driver	No	url, catalog, schema, logging facilities	Application is not portable if using SequeLink specific features.
NonStop JDBC Type 4 Driver version 1.0	No	Potential property value changes (verify using Summary of the Type 4 Driver Properties)	The default file name for logging has changed. For information, see t4LogFile Property .

Third-Party Databases

Applications must be compatible with features and data types supported by SQL/MX. Any application using specific features and data types customized for third-party databases must be modified to work with the Type 4 driver. For features and data types supported by SQL/MX, see the *HPE NonStop SQL/MX Release 3.2.1 Reference Manual*.

Operating Systems

Any NonStop JDBC Type 4 driver application accessing NonStop SQL/MX can be migrated from one supported platform to another (for example, PC to HP-UX) without any application changes.

Compatibility

The NonStop JDBC Type 4 Driver 3.1 is compatible with JDK 1.5 and SQLMX release with BLOB support.

The java 1.6.0_18 or later versions of Oracle enforces the `java.sql.Date.valueOf(String s)` method to follow the date format as `"yyyy-mm-dd"`. The JDBC Type 4 driver throws `illegalArgumentException` for the date format other than `"yyyy-mm-dd"`.

11 Messages

About the Message Format

Messages are listed in numerical SQLCODE order. Descriptions include the following:

```
SQLCODE   SQLSTATE   message-text

Cause [ What occurred to trigger the message.]
Effect [ What is the result when this occurs. ]
Recovery [ How to diagnose and fix the problem. ]
```

Type 4 Driver Error Messages

29001 HYC00 Unsupported feature - {0}

Cause: The feature listed is not supported by the JDBC driver.

Effect: An unsupported exception is throw, and null `resultSet` is returned.

Recovery: Remove the feature functionality from the program.

29002 08003 Connection does not exist

Cause: An action was attempted when the connection to the database was closed.

Effect: The database is inaccessible.

Recovery: Retry the action after the connection to the database is established.

29003 HY000 Statement does not exist

Cause: A validation attempt was made on the getter or exec invocation on a closed statement.

Effect: The getter or exec invocation validation fails.

Recovery: Issue `validateGetInvocation()` or `validateExecDirectInvocation` when the statement is open.

29004 HY024 Invalid transaction isolation value

Cause: An attempt was made to set the transaction isolation level to an invalid value.

Effect: `SQLMXConnection.setTransactionIsolation` does not set the transaction isolation value.

Recovery: Valid isolation values are: `SQL_TXN_READ_COMMITTED`, `SQL_TXN_READ_UNCOMMITTED`, `SQL_TXN_REPEATABLE_READ`, and `SQL_TXN_SERIALIZABLE`. If no isolation value is specified, the default is `SQL_TXN_READ_COMMITTED`.

29005 HY024 Invalid ResultSet type

Cause: An attempt was made to set an invalid `ResultSet` Type value.

Effect: The SQLMXStatement call with the `resultSetType` parameter fails.

Recovery: Valid `ResultSet` types are: `TYPE_FORWARD_ONLY`, `TYPE_SCROLL_INSENSITIVE`, and `TYPE_SCROLL_SENSITIVE`.

29006 HY000 Invalid Result Set concurrency

Cause: An attempt was made to set an invalid result-set concurrency value.

Effect: The SQLMXStatement call with resultSetConcurrency fails.

Recovery: Valid resultSetConcurrency values are: CONCUR_READ_ONLY and CONCUR_UPDATABLE.

29007 07009 Invalid descriptor index

Cause: A ResultSetMetadata column parameter or a ParameterMetaData param parameter is outside of the descriptor range.

Effect: The ResultSetMetadata or ParameterMetaData method data is not returned as expected.

Recovery: Validate the column or parameter that is supplied to the method.

29008 24000 Invalid cursor state

Cause: The ResultSet method was called when the connection was closed.

Effect: The method call does not succeed.

Recovery: Make sure the connection is open before making the ResultSet method call.

29009 HY109 Invalid cursor position

Cause: An attempt was made to perform a deleteRow() method or updateRow() method or cancelRowUpdates method when the ResultSet row cursor was on the insert row. Or an attempt was made to perform the insertRow() method when the ResultSet row cursor was not on the insert row.

Effect: The row changes and cursor manipulation do not succeed.

Recovery: To insert a row, move the cursor to the insert row. To delete, cancel, or update a row, move the cursor from the insert row.

29010 07009 Invalid column name

Cause: A column search does not contain columnName string.

Effect: The column comparison or searches do not succeed.

Recovery: Supply a valid columnName string to the findColumn(), validateGetInvocation(), and validateUpdInvocation() methods.

29011 07009 Invalid column index or descriptor index

Cause: A ResultSet method was issued that has a column parameter that is outside of the valid range.

Effect: The ResultSet method data is not returned as expected.

Recovery: Make sure to validate the column that is supplied to the method.

29012 07006 Restricted data type attribute violation

Cause: An attempt was made to execute a method either while an invalid data type was set or the data type did not match the SQL column type.

Effect: The interface method is not executed.

Recovery: Make sure the correct method and Java data type is used for the column type.

29013 HY024 Fetch size is less than 0

Cause: The size set for ResultSet.setFetchSize rows to fetch is less than zero.

Effect: The number of rows that need to be fetched from the database when more rows are needed for a ResultSet object is not set.

Recovery: Set the `setFetchSize()` method rows parameter to a value greater than zero.

29015 HY024 Invalid fetch direction

Cause: The `setFetchDirection()` method direction parameter is set to an invalid value.

Effect: The direction in which the rows in this `ResultSet` object are processed is not set.

Recovery: Valid fetch directions are: `ResultSet.FETCH_FORWARD`, `ResultSet.FETCH_REVERSE`, and `ResultSet.FETCH_UNKNOWN`.

29017 HY004 SQL data type not supported

Cause: An unsupported `getBytes()` or `setBytes()` JDBC method call was issued using a `BINARY`, `VARBINARY`, or `LONGVARBINARY` data type.

Effect: `BINARY`, `VARBINARY`, and `LONGVARBINARY` data types are not supported.

Recovery: Informational message only; no corrective action is needed.

29018 22018 Invalid character value in cast specification

Cause: An attempt was made to convert a string to a numeric type but the string does not have the appropriate format.

Effect: Strings that are obtained through a getter method cannot be cast to the method type.

Recovery: Validate the string in the database to make sure it is a compatible type.

29019 07002 Parameter {0, number, integer} for {1, number, integer} set of parameters is not set

Cause: An input descriptor contains a parameter that does not have a value set.

Effect: The `method checkIfAllParamsSet()` reports the parameter that is not set.

Recovery: Set a value for the listed parameter.

29020 07009 Invalid parameter index

Cause: A getter or setter method parameter count index is outside of the valid input-descriptor range, or the input-descriptor range is null.

Effect: The getter and setter method invocation validation fails.

Recovery: Change the getter or setter parameter index to a valid parameter value.

29021 HY004 Object type not supported

Cause: A `PreparedStatement.setObject()` method call contains an unsupported Object Type.

Effect: The `setObject()` method does not set a value for the designated parameter.

Recovery: Informational message only; no corrective action is needed. Valid Object Types are: `null`, `BigDecimal`, `Date`, `Time`, `Timestamp`, `Double`, `Float`, `Long`, `Short`, `Byte`, `Boolean`, `String`, `byte[]`, `Clob`, and `Blob`.

29022 HY010 Function sequence error

Cause: The `PreparedStatement.execute()` method does not support the use of the `PreparedStatement.addBatch()` method.

Effect: An exception is reported; the operation is not completed.

Recovery: Use the `PreparedStatement.executeBatch()` method.

29026 HY000 Transaction can't be committed or rolled back when AutoCommit mode is on

Cause: An attempt was made to commit a transaction while `AutoCommit` mode is enabled.

Effect: The transaction is not committed.

Recovery: Disable `AutoCommit`. Use the method only when the `AutoCommit` mode is disabled.

29027 HY011 SetAutoCommit not possible, since a transaction is active

Cause: An attempt was made to call the `setAutoCommit()` mode while a transaction was active.

Effect: The current `AutoCommit` mode is not modified.

Recovery: Complete the transaction, then attempt to set the `AutoCommit` mode.

29029 HY011 SetTransactionIsolation not possible, since a transaction is active

Cause: An attempt was made to set transaction isolation level while a transaction was active.

Effect: Attempts to change the transaction isolation level for this Connection object fail.

Recovery: Complete the transaction, then attempt to set the transaction isolation level.

29031 HY000 SQL SELECT statement in batch is illegal

Cause: A `SELECT` SQL statement was used in the `executeBatch()` method.

Effect: An exception is reported; the `SELECT` SQL query cannot be used in batch queries.

Recovery: Use the `executeQuery()` method to issue the SELECT SQL statement.

29032 23000 Row has been modified since it is last read

Cause: An attempt was made to update or delete a `ResultSet` object row while the cursor was on the insert row.

Effect: The `ResultSet` row modification does not succeed.

Recovery: Move the `ResultSet` object cursor away from the row before updating or deleting the row.

29033 23000 Primary key column value can't be updated

Cause: An attempt was made to update the primary-key column in a table.

Effect: The column is not updated.

Recovery: Columns in the primary-key definition cannot be updated and cannot contain null values, even if you omit the `NOT NULL` clause in the column definition.

29035 HY000 IO Exception occurred {0}

Cause: An ASCII or Binary or Character stream setter or an updater method resulted in a `java.io.IOException`.

Effect: The designated setter or updater method does not modify the ASCII or Binary or Character stream.

Recovery: Informational message only; no corrective action is needed.

29036 HY000 Unsupported encoding {0}

Cause: The character encoding is not supported.

Effect: An exception is thrown when the requested character encoding is not supported.

Recovery: ASCII (ISO88591), KANJI, KSC5601, and UCS2 are the only supported character encodings. SQL/MP tables do not support UCS2 character encoding.

29037 HY106 ResultSet type is TYPE_FORWARD_ONLY

Cause: An attempt was made to point a `ResultSet` cursor to a previous row when the object type is set as `TYPE_FORWARD_ONLY`.

Effect: The `ResultSet` object cursor manipulation does not occur.

Recovery: `TYPE_FORWARD_ONLY ResultSet` object type cursors can move forward only. `TYPE_SCROLL_SENSITIVE` and `TYPE_SCROLL_INSENSITIVE` types are scrollable.

29038 HY107 Row number is not valid

Cause: A `ResultSet absolute()` method was called when the row number was set to 0.

Effect: The cursor is not moved to the specified row number.

Recovery: Supply a positive row number (specifying the row number counting from the beginning of the result set), or supply a negative row number (specifying the row number counting from the end of the result set).

29039 HY092 Concurrency mode of the ResultSet is CONCUR_READ_ONLY

Cause: An action was attempted on a `ResultSet` object that cannot be updated because the concurrency is set to `CONCUR_READ_ONLY`.

Effect: The `ResultSet` object is not modified.

Recovery: For updates, you must set the `ResultSet` object concurrency to `CONCUR_UPDATABLE`.

29040 HY000 Operation invalid. Current row is the insert row

Cause: An attempt was made to retrieve update, delete, or insert information on the current insert row.

Effect: The `ResultSet` row information retrieval does not succeed.

Recovery: To retrieve row information, move the `ResultSet` object cursor away from the insert row.

29041 HY000 Operation invalid. No primary key for the table

Cause: The `getKeyColumns()` method failed on a table that was created without a primary-key column defined.

Effect: No primary-key data is returned for the table.

Recovery: Change the table to include a primary-key column.

29042 HY000 Fetch size value is not valid

Cause: An attempt was made to set the fetch-row size to a value that is less than 0.

Effect: The number of rows that are fetched from the database when more rows are needed is not set.

Recovery: For the `setFetchSize()` method, supply a valid row value that is greater than or equal to 0.

29043 HY000 Max rows value is not valid

Cause: An attempt was made to set a limit of less than 0 for the maximum number of rows that any `ResultSet` object can contain.

Effect: The limit for the maximum number of rows is not set.

Recovery: For the `setMaxRows()` method, use a valid value that is greater than or equal to 0.

29044 HY000 Query timeout value is not valid

Cause: An attempt was made to set a value of less than 0 for the number of seconds the driver waits for a `Statement` object to execute.

Effect: The query timeout limit is not set.

Recovery: For the `setQueryTimeout()` method, supply a valid value that is greater than or equal to 0.

29045 01S07 Fractional truncation

Cause: The data retrieved by the `ResultSet` getter method has been truncated.

Effect: The data retrieved is truncated.

Recovery: Make sure that the data to be retrieved is within a valid data-type range.

29046 22003 Numeric value out of range

Cause: A value retrieved from the `ResultSet` getter method is outside the range for the data type.

Effect: The `ResultSet` getter method does not retrieve the data.

Recovery: Make sure the data to be retrieved is within a valid data-type range.

29047 HY000 Batch update failed. See next exception for details

Cause: One of the commands in a batch update failed to execute properly.

Effect: Not all the batch-update commands succeed. See the subsequent exception for more information.

Recovery: View the subsequent exception for possible recovery actions.

29048 HY009 Invalid use of null

Cause: A parameter that has an expected table name is set to null.

Effect: The `DatabaseMetadata` method does not report any results.

Recovery: For the `DatabaseMetaData` method, supply a valid table name that is not null.

29049 25000 Invalid transaction state

Cause: The `begintransaction()` method was called when a transaction was in progress.

Effect: A new transaction is not started.

Recovery: Before calling the `begintransaction()` method, validate whether other transactions are currently started.

29050 HY107 Row value out of range

Cause: A call to `getCurrentRow` retrieved is outside the first and last row range.

Effect: The current row is not retrieved.

Recovery: It is an informational message only; no recovery is needed. Report the entire message to your service provider.

29051 01S02 ResultSet type changed to TYPE_SCROLL_INSENSITIVE

Cause: The Result Set Type was changed.

Effect: None.

Recovery: This message is reported as an SQL Warning. It is an informational message only; no recovery is needed.

29053 HY000 SQL SELECT statement is invalid in executeUpdate() method

Cause: A select SQL statement was used in the `executeUpdate()` method.

Effect: The `SQL query not performed` exception is reported.

Recovery: Use the `executeQuery()` method to issue the select SQL statement.

29054 HY000 Only SQL SELECT statements are valid in executeQuery() method

Cause: A non-select SQL statement was used in the `executeQuery()` method.

Effect: The exception reported is "`SQL query not performed`".

Recovery: Use the `executeUpdate()` method to issue the non-select SQL statement.

29056 HY000 Statement is already closed

Cause: A `validateSetInvocation()` or `validateExecuteInvocation` method was used on a closed statement.

Effect: The validation on the statement fails and returns an exception.

Recovery: Use the `validateSetInvocation()` or `validateExecuteInvocation` method prior to the statement close.

29057 HY000 Auto generated keys not supported

Cause: An attempt was made to use the Auto-generated keys feature.

Effect: The attempt does not succeed.

Recovery: The Auto-generated keys feature is not supported.

29058 HY000 Connection is not associated with a PooledConnection object

Cause: The `getPooledConnection()` method was invoked before the `PooledConnection` object was established.

Effect: A connection from the pool cannot be retrieved.

Recovery: Make sure a `PooledConnection` object is established before using the `getPooledConnection()` method.

29059 HY000 'blobTableName' property is not set or set to null value or set to invalid value

Cause: Attempted to access a `BLOB` column without setting the property `t4sqlmx.blobTableName`, or the property is set to an invalid value.

Effect: The application cannot access `BLOB` columns.

Recovery: Set the `t4sqlmx.blobTableName` property to a valid LOB table name. The LOB table name is of format *`catalog.schema.lobTableName`*.

29060 HY000 't4sqlmx.clobTableName' property is not set or set to null value or set to invalid value

Cause: Attempted to access a CLOB column without setting the `propertyt4sqlmx.clobTableName` property, or the property is set to null value or set to an invalid value. Effect. The application

Effect: The application cannot access `CLOB` columns.

Recovery: Set the `t4sqlmx.clobTableName` property to a valid LOB table name. The LOB table name is of format *`catalog.schema.lobTableName`*.

29061 HY00 Lob object {0} is not current

Cause: Attempted to access LOB column data after the cursor moved or the result set from which the LOB data was obtained had been closed.

Effect: The application cannot access LOB data.

Recovery: Read the LOB data before moving the cursor or closing the result-set object.

29063 HY00 Transaction error {0} - {1} while obtaining start data locator

Cause: A transaction error occurred when the JDBC/MX driver attempted to reserve the data locators for the given process while inserting or updating a LOB column.

Effect: The application cannot insert or update the LOB columns.

Recovery: Check the file-system error in the message and take recovery action accordingly.

29067 07009 Invalid input value in the method {0}

Cause: One or more input values in the given method is invalid.

Effect: The given input method failed.

Recovery: Check the input values for the given method.

29068 07009 The value for position can be any value between 1 and one more than the length of the LOB data

Cause: The position input value in `Blob.setBinaryStream`, `Clob.setCharacterStream`, or `Clob.setAsciiStream` can be between 1 and one more than the length of the LOB data.

Effect: The application cannot write the LOB data at the specified position.

Recovery: Correct the position input value.

29069 HY000 Autocommit is on and LOB objects are involved.

Cause: An attempt was made to access a LOB column when autocommit made is enabled.

Effect: The application cannot access LOB columns.

Recovery: Disable the autocommit mode.

29100 HY000 An internal error occurred.

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29101 HY000 Contact your Hewlett Packard Enterprise service provider.

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29102 HY000 Error while parsing address *address*

Cause: The address format was not recognized.

Effect: Operation fails.

Recovery: Refer to url Property for the valid address format.

29103 HY000 Address is null

Cause: The address was empty.

Effect: Operation fails.

Recovery: Refer to url Property for the valid address format.

29104 HY000 Expected suffix: *suffix*

Cause: The address suffix was incorrect or missing.

Effect: Operation fails.

Recovery: Refer to url Property for the valid address format.

29105 HY000 Unknown prefix for address

Cause: The address prefix was incorrect or missing.

Effect: Operation fails.

Recovery: Refer to url Property for the valid address format.

29106 HY000 Expected address format: jdbc:subprotocol::subname

Cause: N.A.

Effect: N.A.

Recovery: This is an informational message. Refer to url Property for the valid address format.

29107 HY000 Address not long enough to be a valid address.

Cause: The address length was too short to be a valid address.

Effect: Operation fails.

Recovery: Refer to url Property for the valid address format.

29108 HY000 Expecting \\<machine name><process name>/<port number>.

Cause: The MXCS address format was invalid.

Effect: Operation fails.

Recovery: The address returned by the MXCS association server was not in the expected format. Report the entire message to your service provider.

29109 HY000 //<{IP Address|Machine Name}[:port]/database name>

Cause: Informational message.

Effect: N.A.

Recovery: N.A.

29110 HY000 Address is missing an IP address or machine name.

Cause: An IP address or machine name is required, but missing.

Effect: The operation fails.

Recovery: Include a valid IP address or machine name. Refer to url Property for the valid address format.

29111 HY000 Unable to evaluate address *address* Cause: *cause*

Cause: The driver could not determine the IP address for a host.

Effect: The operation fails.

Recovery: The address or machine name might not be properly qualified or a security restriction might exist. See the documentation for the `getAllByName` method in the `java.net.InetAddress` class. Include a valid IP address or machine name. Refer to url Property for the valid address format.

29112 HY000 Missing ']'.

Cause: The driver could not determine the IP address for a host.

Effect: The operation fails.

Recovery: The address or machine name may not be properly formatted. Refer to [Insert Link to sqlx.htm#url] url for proper address format.

29113 HY000 Error while opening socket. Cause: *cause*

Cause: Socket error.

Effect: The operation fails.

Recovery: Use the `getCause` method on the Exception to determine the appropriate recovery action.

29114 HY000 Error while writing to socket.

Cause: Socket write error.

Effect: The operation fails.

Recovery: Use the `getCause` method on the Exception to determine the appropriate recovery action.

29115 HY000 Error while reading from socket. Cause: *cause*

Cause: Socket read error.

Effect: The operation fails.

Recovery: Use the `getCause` method on the Exception to determine the appropriate recovery action.

29116 HY000 Socket is closed.

Cause: Socket close error.

Effect: The operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29117 HY000 Error while closing session. Cause: *cause*

Cause: An error was encountered while closing a session.

Effect: The operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29118 HY000 A write to a bad map pointer occurred.

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29119 HY000 A write to a bad par pointer occurred.

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29120 HY000 An association server connect message error occurred.

Cause: Unable to connect to the MXCS association server.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29121 HY000 A close message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29122 HY000 An end transaction message error occurred.

Cause: Unable to perform the operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29123 HY000 An execute call message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29124 HY000 An execute direct message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29125 HY000 An execute direct rowset message error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29126 HY000 An execute N message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29127 HY000 An execute rowset message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29128 HY000 A fetch perf message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29129 HY000 A fetch rowset message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29130 HY000 A get sql catalogs message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29131 HY000 An initialize dialogue message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29132 HY000 A prepare message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29133 HY000 A prepare rowset message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29134 HY000 A set connection option message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29135 HY000 A terminate dialogue message error occurred. Cause: *cause*

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29136 HY000 An association server connect reply occurred. Exception: `exception` Exception detail: `exception_detail` Error text/code: `error text_or_code`

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying this message. Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29137 HY000 A close reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29138 HY000 An end transaction reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29139 HY000 An execute call reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29140 HY000 An execute direct reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29141 HY000 An execute direct rowset reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29142 HY000 An execute N reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29143 HY000 An execute rowset reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29144 HY000 A fetch perf reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29145 HY000 A fetch rowset reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29146 HY000 A get sql catalogs reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29147 HY000 An initialize dialogue reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29148 HY000 A prepare reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29149 HY000 A prepare rowset reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29150 HY000 A set connection option reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29151 HY000 A terminate dialogue reply error occurred.

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29152 HY000 No more ports available to start ODBC servers

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: See the *Operator Messages Manual* for server errors. Evaluate the returned value from the `getCause` method on the Exception to determine the appropriate recovery action.

29153 HY000 Invalid authorization specification

Cause: Incorrect user name and/or password.

Effect: Operation fails.

Recovery: Retry with correct user name and/or password.

29154 HY000 Timeout expired

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Retry and/or change the timeout value for the operation.

29155 HY000 Unknown message type

Cause: Internal error.

Effect: Operation fails.

Recovery: Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29156 HY000 An error was returned from the server. Error: `error` Error detail: `error_detail`

Cause: The server reported an error

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29157 HY000 There was a problem reading from the server.

Cause: The server reported an error.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29158 HY000 The message header contained the wrong version. Expected: `expected_version` Actual: `actual_version`

Cause: The server's version differs from the expected version.

Effect: Operation fails.

Recovery: valuate any error or error detail information accompanying the message. Install compatible versions of the driver and MXCS server.

29159 HY000 The message header contained the wrong signature. Expected: `expected_signature` Actual: `actual_signature`

Cause: The server's signature differs from the expected version.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Install compatible versions of the driver and MXCS server.

29160 HY000 The message header was not long enough.

Cause: The message returned by the server was too short to be a valid message.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29161 S1000 Unable to authenticate the user because of an NT error: {0}

Cause: A message returned by the server.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29162 S1000 Unexpected programming exception has been found: exception. Check the server event log on node `node` for details.

Cause: A message returned by the server.

Effect: Operation fails.

Recovery: Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29163 08001 ODBC Services not yet available: *`server`*

Cause: A message returned by the server.

Effect: Operation fails.

Recovery: Retry and/or wait for a server to become available. Configure MXCS server data source with more servers.

29164 08001 DataSource not yet available or not found: *`error`*

Cause: A message returned by the server.

Effect: Operation fails.

Recovery: Create server data source and/or configure server data source with more servers.

29165 HY000 Unknown connect reply error: *error*

Cause: A message returned by the server.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29166 HY000 This method is not implemented.

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29167 HY000 Internal error. An internal index failed consistency check.

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29168 HY000 Unknown reply message error: *error* error detail: error_detail

Cause: Server returned an error.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the *HPE NonStop Connectivity Service Manual for SQL/MX Release 3.2.1* for corrective action.

29169 HY000 Invalid connection property setting

Cause: The message returned by the server was too short to be a valid message.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29170 HY000 Invalid parameter value

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29172 HY000 Translation of parameter to {0} failed.

Cause: Translation errors occurred when translating the parameter into the target character set reported in the *{0}* replacement variable.

Effect: The method fails.

Recovery: Set the parameter to use characters within the appropriate character set. You can also turn off translation validation by setting the translationVerification property to FALSE.

29173 HY000 Translation of SQL statement {0} failed.

Cause: Translation errors occurred when translating the SQL statement into the target character set reported in the *{0}* replacement variable.

Effect: The method fails.

Recovery: Edit the SQL statement to use characters within the appropriate character set. You can also turn off translation validation by setting the `translationVerification` property to `FALSE`.

29174 HY000 Autocommit is on and updateRow was called on the ResultSet object.

Cause: The `ResultSet.updateRow()` method is called when autocommit is set on.

Effect: Warning is thrown. Subsequent `ResultSet.next()` calls will fail because all the `ResultSet(cursors)` are closed.

Recovery: Call the `ResultSet.updateRow()` method with autocommit set to off.

29175 HY000 Unknown Error {0}.

Cause: An unknown error occurred during connection {0}.

Effect: The connection fails.

Recovery: Retry the connection.

29177 HY000 Data cannot be null.

Cause: The data structure returned by the MXCS server contains an unexpected null value.

Effect: Operation fails.

Recovery: Evaluate the EMS message log on the MXCS server, if any, and refer to the SQL/MX Connectivity Service Manual for corrective action.

29178 HY000 No column value has been inserted.

Cause: The value for a required column was not specified.

Effect: Operation fails.

Recovery: Ensure that all required column values are specified, and retry the operation.

01032 08S01 Communication link failure. The server timed out or disappeared.

Cause: The connection timed out.

Effect: Operation fails.

Recovery: Reconnect. Set connection timeout to appropriate value.

01056 25000 Invalid transaction state.

Cause: Transaction state is incorrect.

Effect: Operation fails.

Recovery: Retry.

01118 S1008 Operation cancelled.

Cause: The operation was cancelled.

Effect: Operation fails.

Recovery: Retry operation.

08001 HY000 Retry attempts to connect to the datasource failed, May be ODBC server not able to register to the ODBC service process.

Cause: A server error.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Check the EMS event log for server errors. See the *Operator Messages Manual* for server errors. See the `SQL/MX Connectivity Service Manual` for corrective action.

08004 HY000 Data source rejected establishment of connection since the ODBC server is connected to a different client now

Cause: Connection with server has been lost. Server is now connected to a different connection.

Effect: Operation fails.

Recovery: Reconnect.

29180 HY000 XABROKER returned error. Message: {0}

Cause: Unable to perform the operation.

Effect: Operation fails.

Recovery: None.

29181 HY000 XABROKER returned error. Message: {0}

Cause: Unable to perform the operation.

Effect: Operation fails.

Recovery: None.

29182 HY000 XABROKER max connection limit reached. Broker Message: {0}

Cause: The application attempted to create more connections than the allowed limit.

Effect: The application cannot create more connections to the data source.

Recovery: Evaluate any error or error detail information; check the maximum number of connections that are allowed.

29183 HY000 Invalid XAFLAG values sent to the XABROKER. Broker Message: {0}

Cause: An invalid flag sent to the XABROKER for a transaction operation.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information; use a proper flag for the attempted operation.

29184 HY000 Version mismatch between the Type 4 driver and the XABROKER. Broker Message: {0}

Cause: The XABROKER expects a different version of the driver.

Effect: Operation fails.

Recovery: Evaluate any error or error detail information accompanying the message. Install compatible versions of the driver and XABROKER.

29185 HY000T4 connection pooling is not supported for XA connections. Set T4Property maxPoolSize to -1. {0}

Cause: The feature listed is not supported by the JDBC driver.

Effect: Operation fails.

Recovery: Set the T4 driver maxPoolSize property to -1.

29186 HY000 Internal error occurred while getting the SQLMXXAConnection object.- {0}

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29187 HY000 Internal error occurred while getting the SQLMXXAResource object. - {0}

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Report the entire message to your service provider.

29188 HY000 Invalid Tx connection. Connection closed.- {0}

Cause: An action was attempted when the connection to the database was closed.

Effect: The database is inaccessible.

Recovery: Retry the action after the connection to the database is established.

29189 HY000 Invalid Tx operation on XA connection.- {0}

Cause: The feature listed is not supported by the JDBC driver.

Effect: An attempt was made to perform an invalid Tx operation.

Recovery: Ensure that a proper Transaction operation is performed.

29190 HY000 Invalid XA Connection. - {0}

Cause: There is no connection to the server.

Effect: The operation fails.

Recovery: Verify all the properties to the database are valid.

S1000 HY000 A TIP transaction error has been detected. Check the server event log on Node node for Transaction Error details.

Cause: A message was returned by the server.

Effect: Operation fails.

Recovery: Check EMS event log on node.

29191 HY000 SQL Server and connection aborted successfully (`closeConnectionUponQueryTimeout` Enabled).

Cause: The long running query timed out after specified number of seconds.

Effect: The MXOSRVR on NSK is stopped successfully.

Recovery: None. The Connection object and its child objects are rendered invalid. Re-connect in the Java program.

29192 HY000 Server and connection could not be aborted successfully, PROGRAM ERROR (`closeConnectionUponQueryTimeout` Enabled).

Cause: The long running query timed out after specified number of seconds.

Effect: The MXOSRVR on NSK did not stop successfully.

Recovery: None. This is an internal Error. Marshaling parameters error encountered while communicating with server.

29193 HY000 Server and connection could not be aborted successfully, AS NOT AVAILABLE ERROR (`closeConnectionUponQueryTimeout` Enabled).

Cause: The long running query timed out after specified number of seconds.

Effect: The MXOSRVR on NSK did not stop successfully.

Recovery: Ensure that the AS server is available.

29194 HY000 SQL Server and connection could not be aborted successfully, SERVER NOT FOUND ERROR (`closeConnectionUponQueryTimeout` Enabled).

Cause: The long running query timed out after specified number of seconds.

Effect: The MXOSRVR on NSK could did not stop successfully.

Recovery: None. The corresponding MXOSRVR to be stopped is unavailable.

29195 HY000 SQL Server and connection could not be aborted successfully, SERVER IN USE BY ANOTHER CLIENT ERROR (`closeConnectionUponQueryTimeout` Enabled).

Cause: The long running query timed out after specified number of seconds.

Effect: The MXOSRVR on NSK did not stop successfully.

Recovery: None. The corresponding MXOSRVR is in use by another client.

29196 HY000 SQL Server and connection could not be aborted successfully, SERVER STOP ERROR (`closeConnectionUponQueryTimeout` Enabled).

Cause: The long running query timed out after specified number of seconds.

Effect: The MXOSRVR on NSK did not stop successfully.

Recovery: None. The corresponding MXOSRVR could not be stopped due to an unrecognized error.

29197 HY000 Provided DATE or TIME or TIMESTAMP is not valid and cannot be converted.

Cause: The provided DATE or TIME or TIMESTAMP value is not valid.

Effect: The operation fails.

Recovery: Provide a valid DATE or TIME or TIMESTAMP value.

29198 HY000 Maximum connection pool size is reached.

Cause: Maximum connection pool size is reached.

Effect: An exception is reported; the operation is not completed.

Recovery: Increase the connection pool size.

29199 HY000 TMF error while obtaining transaction handle.

Cause: TMF error while obtaining transaction handle.

Effect: An exception is reported; the operation is not completed.

Recovery: None.

29200 HY000 Internal error while obtaining transaction handle

Cause: Internal error while obtaining transaction handle.

Effect: An exception is reported; the operation is not completed.

Recovery: None.

12 Providing a secure JDBC connection using NonStop SSL

This chapter discusses JDBC connection encryption by using NonStop SSL. This chapter includes the following topics:

Encrypting the JDBC connection

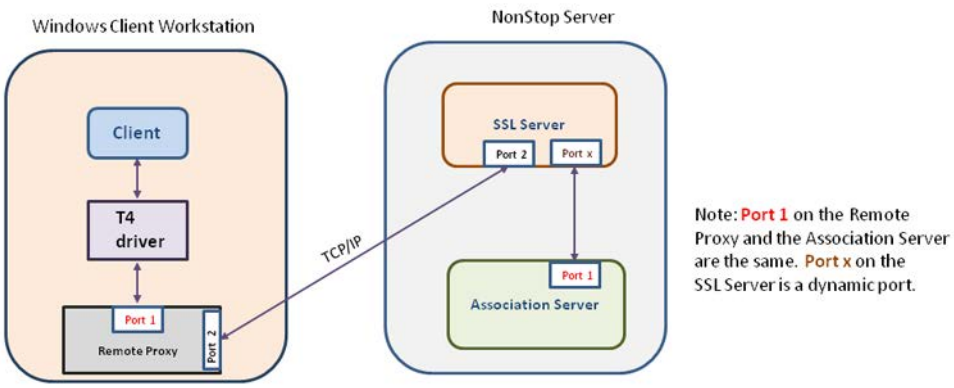
To encrypt the JDBC connection by using NonStop SSL, complete the following tasks:

1. On the HPE NonStop server, install a NonStop SSL Server process for the ODBC/MX Association Server of the target MXCS subsystem.
2. On each workstation, install the RemoteProxy software. Next, configure each Remote Proxy to route plain ODBC and JDBC connections to the NonStop SSL Server process installed in step 1.
3. To enable the JDBC Type 4 driver on your workstation to connect to RemoteProxy, on each workstation, modify the connection string and the properties file of the appropriate Java applications.

Secure JDBC connection architecture

Figure 4 (page 112) describes the architecture of the secure JDBC connection.

Figure 4 Secure JDBC connection



Installing a NonStop SSL Server process for ODBC/MX

Installing a NonStop SSL Server process is a privileged task that is restricted to the `super.super` user. Users configuring only client workstations can ignore the following section that summarizes the NonStop SSL installation process.

To install a NonStop SSL Server process for ODBC/MX, complete the following steps on the NonStop server:

1. Log on as `super.super`.
2. To change to the NonStop SSL directory:

```
$SYSTEM STARTUP 1> volume $system.znsssl
```

3. To run the NonStop SSL SETUP, enter the following command at the TACL prompt:

   ```
   $SYSTEM ZNSSSL 10> run $SYSTEM.ZNSSSL.SETUP
   ```

4. Enter [7] to select ODBC/MX SERVER as run mode.
5. Enter the home terminal. The default value is `$YMIOP.#CLCI`.
6. Enter the CPU on which you want to run SSL. The default value is CPU 3.
7. Enter the SSL process name. The default value is `$ODBS3`.
8. Enter the TCP/IP process name for the subnet on which the ODBC/MX MXCS Service runs. The default value is `$ZTC0`.
9. Enter a port number for the SSL ODBC/MX connection. The default port number is 8402.
10. Enter y or n to specify whether the startup and error messages must be sent to EMS.
11. Enter a name for the `SCF IN` file for the ODBCMXS configuration. The default value is `ODBSIN3`.
12. Enter a name for the `SCF` configuration file for the ODBCMXS configuration. The default value is `ODBSCF3`.

    The following message shows that SSL configuration has successfully created the script files that are to be used to configure and start the process as a kernel managed persistent process:

    ```
    **** SETUP has completed successfully ****

    Files created:  <SCF IN file name> and <SCF configuration file>.
    ```

13. Enter the following command to configure NonStop SSL as a persistent process:

    ```
    $SYSTEM ZNSSSL 11> SCF/IN <SCF IN file name>
    ```

    For example, if you use the default name, the command is **`SCF/IN ODBSIN3/`**

14. Enter the following command to start the process:

    ```
    $SYSTEM ZNSSSL 12> SCF START PROCESS $ZZKRN.#SSL-ODBCMXS-3
    ```

15. Enter the following command to check whether the ODBCMXS process has started correctly:

    ```
    $SYSTEM ZNSSSL 13> SHOWLOG ODBSLOG *
    ```

    The log file must contain a message similar to the following sample:

    ```
     $ODBS3|05Mar12 09:54:19.89|20|ODBC/MX server proxy started on target host 127.0.0.1, target port 23,
    source port 8402 $ODBS3|05Mar12 09:54:19.93|50|Performed action
    CombinedAction(CreateSocket,ListenSocketAction) successfully.
    ```

    The host NonStop SSL Server process is now running and available for client access.

For more information on installing the NonStop SSL Server process for ODBC/MX, see the *HPE NonStop SSL Reference Manual*.

Installing and configuring the Remote Proxy Client on the Windows workstation

You must obtain the following information from your administrator before installing the Remote Proxy Client:

- The IP address, host name, and port number of the SSL server running on the NonStop system.
- The port number of the MXCS Association Server that you want to connect to, on the NonStop system.

NOTE: The MXCS Association Server port number must not be in use by any other program or service on your client workstation.

For information on obtaining the port number, see the *SQL/MX Connectivity Service Administrative Command Reference*.

To install and configure the Remote Proxy Client on Windows, complete the following steps on the Windows workstation:

1. Download the `$SYSTEM.ZNSSSL.PROXYEXE` file in binary format to your ODBC/MX client workstation, and rename it to `PROXY.EXE`.
2. On the ODBC/MX client workstation, run `PROXY.EXE` to start the RemoteProxy installation program, and follow the installation instructions in the wizards.
3. Double-click the NonStop SSL RemoteProxy icon in your system tray.

   The RemoteProxy configuration window is displayed.
4. Select **Session→New**.

   The **HP NonStop SSL RemoteProxy** dialog is displayed.
   a. In the **Protocol** menu, select **ODBC/MX Client**.
   b. In the **Target (Connecting) Host** field, enter the IP address or host name where your ODBCMXS process is listening on the HPE NonStop server.
   c. In the **Target (Connecting) Port** field, enter the port number of the SSL server. For example, 8402.
   d. In the **Local (Accepting) Port** field, enter the port number of the MXCS Association Server.

   NOTE: The MXCS Association Server port number must not be in use by any other program or service on your client workstation. The port number considered in this example is 18888.
5. Click **Start** to start the Remote Proxy session.
6. Check the startup messages to verify that the Remote Proxy session has started successfully.

Modifying the connection string and properties file of the JDBC Type 4 driver

To modify the connection string on your workstation to connect to RemoteProxy, complete the following steps:

1. Route the connections from the JDBC Type 4 driver through RemoteProxy by updating the connection string of the Java application. Modify the URL as shown in the following example:

   ```
   c = DriverManager.getConnection("jdbc:t4sqlmx://localhost:18888/:serverDataSource=LL1;", "user", "password");
   ```

   `LL1` is a sample server-side data source used for this example.

   For more information on connecting to the server-side data source using the specified URL, see the “Accessing SQL Databases with SQL/MX” (page 21).
2. Run the application.

   If errors are generated, see the server and client SSL logs.

   For information on modifying the properties file, see “Specifying JDBC Type 4 Properties” (page 39).

13 Support and other resources

Accessing Hewlett Packard Enterprise Support

- For live assistance, go to the Contact Hewlett Packard Enterprise Worldwide website:
  www.hpe.com/assistance
- To access documentation and support services, go to the Hewlett Packard Enterprise Support Center website:
  www.hpe.com/support/hpesc

Information to collect

- Technical support registration number (if applicable)
- Product name, model or version, and serial number
- Operating system name and version
- Firmware version
- Error messages
- Product-specific reports and logs
- Add-on products or components
- Third-party products or components

Accessing updates

- Some software products provide a mechanism for accessing software updates through the product interface. Review your product documentation to identify the recommended software update method.
- To download product updates, go to either of the following:
  - Hewlett Packard Enterprise Support Center **Get connected with updates** page:
    www.hpe.com/support/e-updates
  - Software Depot website:
    www.hpe.com/support/softwaredepot
- To view and update your entitlements, and to link your contracts and warranties with your profile, go to the Hewlett Packard Enterprise Support Center **More Information on Access to Support Materials** page:
  www.hpe.com/support/AccessToSupportMaterials

IMPORTANT: Access to some updates might require product entitlement when accessed through the Hewlett Packard Enterprise Support Center. You must have an HP Passport set up with relevant entitlements.

Websites

Website	Link
Hewlett Packard Enterprise Information Library	www.hpe.com/info/enterprise/docs
Hewlett Packard Enterprise Support Center	www.hpe.com/support/hpesc

Website	Link
Contact Hewlett Packard Enterprise Worldwide	www.hpe.com/assistance
Subscription Service/Support Alerts	www.hpe.com/support/e-updates
Software Depot	www.hpe.com/support/softwaredepot
Customer Self Repair	www.hpe.com/support/selfrepair
Insight Remote Support	www.hpe.com/info/insightremotesupport/docs
Serviceguard Solutions for HP-UX	www.hpe.com/info/hpux-serviceguard-docs
Single Point of Connectivity Knowledge (SPOCK) Storage compatibility matrix	www.hpe.com/storage/spock
Storage white papers and analyst reports	www.hpe.com/storage/whitepapers

Customer self repair

Hewlett Packard Enterprise customer self repair (CSR) programs allow you to repair your product. If a CSR part needs to be replaced, it will be shipped directly to you so that you can install it at your convenience. Some parts do not qualify for CSR. Your Hewlett Packard Enterprise authorized service provider will determine whether a repair can be accomplished by CSR.

For more information about CSR, contact your local service provider or go to the CSR website:

www.hpe.com/support/selfrepair

Remote support

Remote support is available with supported devices as part of your warranty or contractual support agreement. It provides intelligent event diagnosis, and automatic, secure submission of hardware event notifications to Hewlett Packard Enterprise, which will initiate a fast and accurate resolution based on your product's service level. Hewlett Packard Enterprise strongly recommends that you register your device for remote support.

For more information and device support details, go to the following website:

www.hpe.com/info/insightremotesupport/docs

Documentation feedback

Hewlett Packard Enterprise is committed to providing documentation that meets your needs. To help us improve the documentation, send any errors, suggestions, or comments to Documentation Feedback (**docsfeedback@hpe.com**). When submitting your feedback, include the document title, part number, edition, and publication date located on the front cover of the document. For online help content, include the product name, product version, help edition, and publication date located on the legal notices page.

A Sample Programs Accessing CLOB and BLOB Data

This appendix shows two working programs.

Sample Program Accessing CLOB Data

This sample program shows operations that can be performed through the Clob interface or through the `PreparedStatement` interface. The sample program shows examples of both interfaces taking a variable and putting the variable's value into a base table that has a `CLOB` column.

```
// LOB operations can be performed through the Clob interface,
// or the PreparedStatement interface.
// This program shows examples of both interfaces taking a
// variable and putting it into the cat.sch.clobbase table.
//
// The LOB base table for this example is created as:
//   >> create table clobbase
//     (col1 int not null not droppable,
//     col2 clob, primary key (col1));
//
// The LOB table for this example is created through
// the T4LobAdmin utility as:
//   >> create table cat.sch.clobdatatbl
//     (table_name char(128) not null not droppable,
//     data_locator largeint not null not droppable,
//     chunk_no int not null not droppable,
//     lob_data varchar(3880),
//     primary key(table_name, data_locator, chunk_no))
//     attributes extent(1024), maxextents 768 ;
//
// ***** The following is the Clob interface...
//   - insert the base row with EMPTY_CLOB() as value for
//   the LOB column
//   - select the LOB column 'for update'
//   - load up a byte[] with the data
//   - use Outputstream.write(byte[])
//
// ***** The following is the PreparedStatement interface...
//   - need an Inputstream object that already has data
//   - need a PreparedStatement object that contains the
//   'insert...' DML of the base table
//   - ps.setAsciiStream() for the lob data
//   - ps.executeupdate(); for the DML
//
// To run this example, issue the following:
//   # java TestCLOB 1 TestCLOB.java 1000
//

import java.sql.*;
import java.io.*;

public class TestCLOB
{
public static void main (String[] args)
    throws java.io.FileNotFoundException,
    java.io.IOException
{
int    length = 500;
int    recKey;
long    start;
```

```
long    end;
Connection    conn1 = null;

// Set t4sqlmx.clobTableName System Property. This property
// can also be added to the command line through
// "-Dt4sqlmx.clobTableName=...", or a
// java.util.Properties object can be used and passed to
// getConnection.
System.setProperty( "t4sqlmx.clobTableName","cat.sch.clobdatatbl" );

if (args.length < 2) {
    System.out.println("arg[0]=; arg[1]=file;
    arg[2]=");
    return;
}
String k = "K";
for (int i=0; i<5000; i++) k = k + "K";
System.out.println("string length = " + k.length());

FileInputStream clobFs = new FileInputStream(args[1]);
int clobFsLen = clobFs.available();

if (args.length == 3)
    length = Integer.parseInt(args[2]);
recKey = Integer.parseInt(args[0]);

System.out.println("Key: " + recKey +"; Using "
+ length + " of file " + args[1]);

try {
 Class.forName("com.tandem.t4jdbc.SQLMXDriver");
 start = System.currentTimeMillis();
 //url should be of the form:
 // jdbc:t4sqlmx://ip_address|machine_name:port_number/:”
 String url = “jdbc:t4sqlmx://mymachine:6000/:”;
 conn1 = DriverManager.getConnection(url);

 System.out.println("Cleaning up test tables...");
 Statement stmt0 = conn1.createStatement();
 stmt0.execute("delete from clobdatatbl");
 stmt0.execute("delete from clobbase");
 conn1.setAutoCommit(false);

 }
 catch (Exception e1) {
 e1.printStackTrace();
 }

// PreparedStatement interface example - This technique
// is suitable if the LOB data is already on the NonStop
// system disk.
 try {
  System.out.println("PreparedStatement interface
  LOB insert..."); String stmtSource1 = "insert into clobbase
  values (?,?)";
  PreparedStatement stmt1
  = conn1.prepareStatement(stmtSource1);
  stmt1.setInt(1,recKey);
  stmt1.setAsciiStream(2,clobFs,length);
  stmt1.executeUpdate();
  conn1.commit();
  }
catch (SQLException e) {
 e.printStackTrace();
 SQLException next = e;
```

```
 do {
    System.out.println("Messge : " + e.getMessage());
    System.out.println("Error Code : " + e.getErrorCode());
    System.out.println("SQLState : " + e.getSQLState());
  } while ((next = next.getNextException()) != null);
}

// Clob interface example - This technique is suitable when
// the LOB data is already in the app, such as having been
// transferred in a msgbuf.
try {
   // insert a second base table row with an empty LOB column
   System.out.println("CLOB interface EMPTY LOB insert...");
   String stmtSource2 = "insert into clobbase
   values (?,EMPTY_CLOB())";
   PreparedStatement stmt2
   = conn1.prepareStatement(stmtSource2);
   stmt2.setInt(1,recKey+1);
   stmt2.executeUpdate();

   Clob clob = null;

   System.out.println("Obtaining CLOB data to
   update (EMPTY in this case)...");
   PreparedStatement stmt3
   = conn1.prepareStatement("select col2
   from clobbase where col1 = ? for update");
   stmt3.setInt(1,recKey+1);
   ResultSet rs = stmt3.executeQuery();
   if (rs.next()) clob = rs.getClob(1); // has to be there
//           else the base table insert fails
   System.out.println("Writing data to previously empty CLOB...");
   OutputStream os = clob.setAsciiStream(1);
   byte[] bData = k.getBytes();
   os.write(bData);
   os.close();
   conn1.commit();
}
catch (SQLException e) {
   e.printStackTrace();
   SQLException next = e;
   do {
    System.out.println("Messge : " + e.getMessage());
    System.out.println("Vendor Code : " + e.getErrorCode());
    System.out.println("SQLState : " + e.getSQLState());
   } while ((next = next.getNextException()) != null);
  }
 } // main
} // class
```

Sample Program Accessing BLOB Data

This sample program shows the use of both the Blob interface and the `PreparedStatement` interface to take a byte variable and put the variable's value into a base table that has a `BLOB` column.

```
// LOB operations may be performed through the Blob, or
// PreparedStatement interface. This program shows examples of
// using both interfaces taking a byte[] variable and putting
// it into the cat.sch.blobtiff table.
//
// The LOB base table for this example is created as:
// >> create table blobtiff
// (col1 int not null not droppable,
// tiff blob, primary key (col1));
```

```
//
// The LOB table for this example is created through the
// T4LobAdmin utility as:
// >> create table cat.sch.blobdatatbl
// (table_name char(128) not null not droppable,
// data_locator largeint not null not droppable,
// chunk_no int not null not droppable,
// lob_data varchar(3880),
// primary key(table_name, data_locator, chunk_no))
// attributes extent(1024), maxextents 768 ;
//
// ***** The following is the blob interface...
// - insert the base row with EMPTY_BLOB() as value for
// the LOB column
// - select the lob column 'for update'
// - load up a byte[] with the data
// - use Outputstream.write(byte[])
//
// ***** The following is the prep stmt interface...
// - need an Inputstream object that already has data
// - need a PreparedStatement object that contains the
// 'insert...' DML of the base table
// - ps.setAsciiStream() for the lob data
// - ps.executeupdate(); for the DML
//
// To run this example, issue the following:
// # java TestBLOB 1 TestBLOB.class 1000
//
import java.sql.*;
import java.io.*;

public class TestBLOB
{
 public static void main (String[] args)
   throws java.io.FileNotFoundException, java.io.IOException
{
int   numBytes;
int   recKey;
long   start;
long   end;
Connection conn1 = null;
//

Set t4slqmx.blobTableName System Property. This property
// can also be added to the command line through
// "-Dt4slqmx.blobTableName=...", or a
// java.util.Properties object can be used and passed to
// getConnection.
System.setProperty( "t4slqmx.blobTableName","cat.sch.blobdatatbl" );

if (args.length < 2) {
   System.out.println("arg[0]=; arg[1]=file; arg[2]=");
   return;
}

// byte array for the blob
byte[] whatever = new byte[5000];
for (int i=0; i<5000; i++) whatever[i] = 71; // "G"

String k = "K";
for (int i=0; i<5000; i++) k = k + "K";
System.out.println("string length = " + k.length());

java.io.ByteArrayInputStream iXstream
  = new java.io.ByteArrayInputStream(whatever);
```

```
numBytes = iXstream.available();
if (args.length == 3)
   numBytes = Integer.parseInt(args[2]);
recKey = Integer.parseInt(args[0]);

System.out.println("Key: " + recKey +"; Using "
     + numBytes + " of file " + args[1]);

try {
  Class.forName("com.tandem.t4jdbc.SQLMXDriver");
  start = System.currentTimeMillis();

  //url should be of the form:
  // jdbc:t4sqlmx://ip_address|machine_name:port_number/:
  String url = "jdbc:t4sqlmx://mymachine:port/:";
  String user ="UserName";
  String password="password";
  conn1 = DriverManager.getConnection(url,user,password);

  System.out.println("Cleaning up test tables...");
  Statement stmt0 = conn1.createStatement();
  stmt0.execute("delete from blobdatatbl");
  stmt0.execute("delete from blobtiff");

  conn1.setAutoCommit(false);

} catch (Exception e1) {
  e1.printStackTrace();
}
// PreparedStatement interface example - This technique is
// suitable if the LOB data is already on the
// NonStop system disk.
 try {
  System.out.println("PreparedStatement interface LOB insert...");
  String stmtSource1 = "insert into blobtiff values (?,?)";
  PreparedStatement stmt1 = conn1.prepareStatement(stmtSource1);
  stmt1.setInt(1,recKey);
  stmt1.setBinaryStream(2,iXstream,numBytes);
  stmt1.executeUpdate();
  conn1.commit();
 }
 catch (SQLException e) {
  e.printStackTrace();
  SQLException next = e;
  do {
   System.out.println("Messge : " + e.getMessage());
   System.out.println("Error Code : " + e.getErrorCode());
   System.out.println("SQLState : " + e.getSQLState());
  } while ((next = next.getNextException()) != null);
 }
// Blob interface example - This technique is suitable when
// the LOB data is already in the app, such as having been
// transfered in a msgbuf.
 try {
  // insert a second base table row with empty LOB column
  System.out.println("BLOB interface LOB insert...");
  String stmtSource2 = "insert into blobtiff
  values (?,EMPTY_BLOB())";
  PreparedStatement stmt2 = conn1.prepareStatement(stmtSource2);
  stmt2.setInt(1,recKey+1);
  stmt2.executeUpdate();

  Blob tiff = null;
```

```
  System.out.println("Obtaining BLOB data to
  update (EMPTY in this case)...");
  PreparedStatement stmt3 = conn1.prepareStatement("select tiff
  from blobtiff where col1 = ? for update");
  stmt3.setInt(1,recKey+1);
  ResultSet rs = stmt3.executeQuery();
  if (rs.next()) tiff = rs.getBlob(1); // has to be there
  else the base table insert failed

  System.out.println("Writing data to previously
  empty BLOB...");
  OutputStream os = tiff.setBinaryStream(1);
  byte[] bData = k.getBytes();
  os.write(bData); os.close();
  conn1.commit();
 }
 catch (SQLException e) {
  e.printStackTrace();
  SQLException next = e;
  do {
   System.out.println("Messge : " + e.getMessage());
   System.out.println("Vendor Code : " + e.getErrorCode());
   System.out.println("SQLState : " + e.getSQLState());
  } while ((next = next.getNextException()) != null);
 }
} // main
} // class
```

B Warranty and regulatory information

For important safety, environmental, and regulatory information, see *Safety and Compliance Information for Server, Storage, Power, Networking, and Rack Products*, available at www.hpe.com/support/Safety-Compliance-EnterpriseProducts.

Warranty information

HPE ProLiant and x86 Servers and Options

www.hpe.com/support/ProLiantServers-Warranties

HPE Enterprise Servers

www.hpe.com/support/EnterpriseServers-Warranties

HPE Storage Products

www.hpe.com/support/Storage-Warranties

HPE Networking Products

www.hpe.com/support/Networking-Warranties

Regulatory information

Belarus Kazakhstan Russia marking

Manufacturer and Local Representative Information

Manufacturer information:

- Hewlett Packard Enterprise Company, 3000 Hanover Street, Palo Alto, CA 94304 U.S.

Local representative information Russian:

- **Russia**:

  ООО «Хьюлетт Паккард Энтерпрайз», Российская Федерация, 125171, г. Москва, Ленинградское шоссе, 16А, стр.3, Телефон/факс: +7 495 797 35 00

- **Belarus**:

  ИООО «Хьюлетт-Паккард Бел», Республика Беларусь, 220030, г. Минск, ул. Интернациональная, 36-1, Телефон/факс: +375 17 392 28 20

- **Kazakhstan**:

  ТОО «Хьюлетт-Паккард (К)», Республика Казахстан, 050040, г. Алматы, Бостандыкский район, проспект Аль-Фараби, 77/7, Телефон/факс: + 7 727 355 35 52

Local representative information Kazakh:

- **Russia**:

  ЖШС "Хьюлетт Паккард Энтерпрайз", Ресей Федерациясы, 125171,
  Мәскеу, Ленинград тас жолы, 16А блок 3, Телефон/факс: +7 495 797 35 00

- **Belarus**:

  «HEWLETT-PACKARD Bel» ЖШС, Беларусь Республикасы, 220030, Минск қ.,
  Интернациональная көшесі, 36/1, Телефон/факс: +375 17 392 28 20

- **Kazakhstan**:

  ЖШС «Хьюлетт-Паккард (К)», Қазақстан Республикасы, 050040, Алматы қ.,
  Бостандық ауданы, Әл-Фараби даңғ ылы, 77/7, Телефон/факс: +7 727 355 35 52

Manufacturing date:

The manufacturing date is defined by the serial number.

CCSYWWZZZZ (serial number format for this product)

Valid date formats include:

- YWW, where Y indicates the year counting from within each new decade, with 2000 as the starting point; for example, 238: 2 for 2002 and 38 for the week of September 9. In addition, 2010 is indicated by 0, 2011 by 1, 2012 by 2, 2013 by 3, and so forth.
- YYWW, where YY indicates the year, using a base year of 2000; for example, 0238: 02 for 2002 and 38 for the week of September 9.

Turkey RoHS material content declaration

Türkiye Cumhuriyeti: EEE Yönetmeliğine Uygundur

Ukraine RoHS material content declaration

Обладнання відповідає вимогам Технічного регламенту щодо обмеження використання деяких небезпечних речовин в електричному та електронному обладнанні, затвердженого постановою Кабінету Міністрів України від 3 грудня 2008 № 1057

Glossary

A

abstract class In Java, a class designed only as a parent from which subclasses can be derived, which is not itself suitable for instantiation. An abstract class is often used to "abstract out" incomplete sets of features, which can then be shared by a group of sibling subclasses that add different variations of the missing pieces.

American National Standards Institute (ANSI) The United States government body responsible for approving US standards in many areas, including computers and communications. ANSI is a member of ISO. ANSI sells ANSI and ISO (international) standards.

American Standard Code for Information Interchange (ASCII) The predominant character set encoding of present-day computers. ASCII uses 7 bits for each character. It does not include accented letters or any other letter forms not used in English (such as the German sharp-S or the Norwegian ae-ligature). Compare to Unicode.

ANSI See American National Standards Institute (ANSI).

API See application program interface (API).

application program
1. A software program written for or by a user for a specific purpose.
2. A computer program that performs a data processing function rather than a control function.

application program interface (API) A set of functions or procedures that are called by an application program to communicate with other software components.

ASCII See American Standard Code for Information Interchange (ASCII).

autocommit mode A mode in which a JDBC driver automatically commits a transaction without the programmer's calling `commit()`.

AWT See Abstract Window Toolkit (AWT).

B

base table A table that has physical existence: that is, a table stored in a file.

BLOB Short for Binary Large OBject, a collection of binary data stored as a single entity in a database management system. These entities are primarily used to hold multimedia objects such as images, videos, and sound. They can also be used to store programs or even fragments of code. A Java `Blob` object (Java type, `java.sql.Blob`) corresponds to the SQL `BLOB` data type.

branded A Java virtual machine that Oracle has certified as conformant.

browser A program that allows you to read hypertext. The browser gives some means of viewing the contents of nodes and of navigating from one node to another. Internet Explorer, Netscape Navigator, NCSA Mosaic, Lynx, and W3 are examples for browsers for the WWW. They act as clients to remote servers.

bytecode The code that javac, the Java compiler, produces.

C

catalog In SQL/MP and SQL/MX, a set of tables containing the descriptions of SQL objects such as tables, views, columns, indexes, files, and partitions.

class path The directories where a Java virtual machine and other Java programs that are located in the `/usr/tandem/java/bin` directory search for class libraries (such as classes.zip). You can set the class path explicitly or with the `CLASSPATH` environment variable.

client A software process, hardware device, or combination of the two that requests services from a server. Often, the client is a process residing on a programmable workstation and is the part

of a program that provides the user interface. The workstation client might also perform other portions of the program logic. Also called a requester.

CLOB Short for Character Large OBject, text data stored as a single entity in a database management system. A Java `Clob` object (Java type, `java.sql.Clob`) corresponds to the SQL `CLOB` data type.

command The operation demanded by an operator or program; a demand for action by, or information from, a subsystem. A command is typically conveyed as an interprocess message from a program to a subsystem.

concurrency A condition in which two or more transactions act on the same record in a database at the same time. To process a transaction, a program must assume that its input from the database is consistent, regardless of any concurrent changes being made to the database. TMF manages concurrent transactions through concurrency control.

concurrency control Protection of a database record from concurrent access by more than one process. TMF imposes this control by dynamically locking and unlocking affected records to ensure that only one transaction at a time accesses those records.

connection pooling A framework for pooling JDBC connections.

Core Packages The required set of APIs in a Java platform edition which must be supported in any and all compatible implementations.

D

Data Control Language (DCL) The set of data control statements within the SQL language.

Data Manipulation Language (DML) The set of data-manipulation statements within the SQL/MP language. These statements include INSERT, DELETE, and UPDATE, which cause database modifications that Remote Duplicate Database Facility (RDF) can replicate.

DCL See Data Control Language (DCL).

DML See Data Manipulation Language (DML).

driver A class in JDBC that implements a connection to a particular database management system such as NonStop SQL/MX. The NonStop Server for Java 6.0 has these driver implementations: JDBC Driver for SQL/MP(JDBC/MP) and JDBC Driver for SQL/MX (JDBC/MX).

`DriverManager` The JDBC class that manages drivers.

E

exception An event during program execution that prevents the program from continuing normally; generally, an error. Java methods raise exceptions using the throw keyword and handle exceptions using try, catch, and finally blocks.

Expand The HPE NonStop operating system network that extends the concept of fault tolerance to networks of geographically distributed NonStop systems. If the network is properly designed, communication paths are constantly available even if there is a single line failure or component failure.

expandability See scalability.

F

fault tolerance The ability of a computer system to continue processing during and after a single fault (the failure of a system component) without the loss of data or function.

G

`get()` method A method used to read a data item. For example, the `SQLMPConnection.getAutoCommit()` method returns the transaction mode of the JDBC driver's connection to an SQL/MP or SQL/MX database. Compare to set() method.

Guardian An environment available for interactive and programmatic use with the NonStop operating system. Processes that run in the Guardian environment use the Guardian system procedure calls as their API. Interactive users of the Guardian environment use the TACL or another Hewlett Packard Enterprise product's command interpreter. Compare to OSS.

GUI See graphical user interface (GUI).

H

Hotspot virtual machine See Java Hotspot virtual machine.

HPE JDBC Driver for SQL/MP (JDBC/MP) The product that provides access to SQL/MP and conforms to the JDBC API.

HPE JDBC Driver for SQL/MX (JDBC/MX) The product that provides access to SQL/MX and conforms to the JDBC API.

HPE NonStop ODBC Server The Hewlett Packard Enterprise implementation of ODBC for NonStop systems.

HPE NonStop operating system The operating system for NonStop systems.

HPE NonStop Server for Java Transaction API An implementation of Java Transaction API (JTA). One version of the NonStop Server for Java Transaction API uses JTS and another uses TMF.

HPE NonStop Server for Java, based on Java Standard Edition 5.0 The formal name of the NonStop Server for Java product whose Java virtual machine conforms to the Java 2 Platform, Standard Edition (J2SE) 5.0. See also NonStop Server for Java 5.

HPE NonStop SQL/MP (SQL/MP) HPE NonStop Structured Query Language/MP, the Hewlett Packard Enterprise relational database management system for NonStop servers.

HPE NonStop SQL/MX (SQL/MX) HPE NonStop Structured Query Language/MX, the Hewlett Packard Enterprise next-generation relational database management system for business-critical applications on NonStop servers.

HPE NonStop system Hewlett Packard Enterprise computers (hardware and software) that support the NonStop operating system.

HPE NonStop Technical Library The browser-based interface to NonStop computing technical information.

HPE NonStop Transaction Management Facility (TMF) A Hewlett Packard Enterprise product that provides transaction protection, database consistency, and database recovery. SQL statements issued through a JDBC driver against a NonStop SQL database call procedures in the TMF subsystem.

HPE NonStop TS/MP (TS/MP) A Hewlett Packard Enterprise product that supports the creation of Pathway servers to access NonStop SQL/MP or Enscribe databases in an online transaction processing (OLTP) environment.

HPE Tandem Advanced Command Language (TACL) The command interpreter for the operating system, which also functions as a programming language, allowing users to define aliases, macros, and function keys.

HTML See Hypertext Markup Language (HTML).

HTTP See Hypertext Transfer Protocol (HTTP).

hyperlink A reference (link) from a point in one hypertext document to a point in another document or another point in the same document. A browser usually displays a hyperlink in a different color,

font, or style. When the user activates the link (usually by clicking on it with the mouse), the browser displays the target of the link.

hypertext A collection of documents (nodes) containing cross-references or links that, with the aid of an interactive browser, allow a reader to move easily from one document to another.

Hypertext Mark-up Language (HTML) A hypertext document format used on the World Wide Web.

Hypertext Transfer Protocol (HTTP) The client - server TCP/IP protocol used on the World Wide Web for the exchange of HTML documents.

I

IEC See International Electrotechnical Commission (IEC).

IEEE Institute for Electrical and Electronic Engineers (IEEE).

inlining Replacing a method call with the code for the called method, eliminating the call.

interactive Question-and-answer exchange between a user and a computer system.

interface In general, the point of communication or interconnection between one person, program, or device and another, or a set of rules for that interaction. See also API.

International Electrotechnical Commission (IEC) A standardization body at the same level as ISO.

International Organization for Standardization (ISO) A voluntary, nontreaty organization founded in 1946, responsible for creating international standards in many areas, including computers and communications. Its members are the national standards organizations of 89 countries, including ANSI.

Internet The network of many thousands of interconnected networks that use the TCP/IP networking communications protocol. It provides e-mail, file transfer, news, remote login, and access to thousands of databases. The Internet includes three kinds of networks:

- High-speed backbone networks such as NSFNET and MILNET
- Mid-level networks such as corporate and university networks
- Stub networks such as individual LANs

Internet Protocol version 6 (IPv6) IP specifies the format of packets and the addressing scheme. The current version of IP is IPv4. IPv6 is a new version of IP designed to allow the Internet to grow steadily, both in terms of number of hosts connected and the total amount of data traffic transmitted. (IP is pronounced eye-pea)

interoperability

1. The ability to communicate, execute programs, or transfer data between dissimilar environments, including among systems from multiple vendors or with multiple versions of operating systems from the same vendor. Hewlett Packard Enterprise documents often use the term connectivity in this context, while other vendors use *connectivity* to mean hardware compatibility.
2. Within a NonStop system node , the ability to use the features or facilities of one environment from another. For example, the `gtacl` command in the OSS environment allows an interactive user to start and use a Guardian tool in the Guardian environment.

interpreter The component of a Java virtual machine that interprets bytecode into native machine code.

Invocation API The C-language API that starts a Java virtual machine and invokes methods on it. The Invocation API is a subset of the JNI.

IPv6 See Internet Protocol version 6 (IPv6).

ISO See International Organization for Standardization (ISO).

iTP Secure WebServer	The Hewlett Packard Enterprise web server with which the NonStop Server for Java integrates using servlets.

J

`jar`	The Java Archive tool, which combines multiple files into a single Java Archive (JAR) file. Also, the command to run the Java Archive Tool.
JAR file	A Java Archive file, produced by the Java Archive Tool, jar.
`java`	The Java application launcher, which launches an application by starting a Java runtime environment, loading a specified class, and invoking that class's `main` method.
Java 2 Platform, Enterprise Edition (J2EE platform)	An environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs) and protocols that provide the functionality for developing multi-tiered, Web-based applications.
Java Conformance Kit (JCK)	The collection of conformance tests that any vendor's JDK must pass in order to be conformant with the Oracle specification.
Java Database Connectivity (JDBC)	An industry standard for database-independent connectivity between the Java platform and relational databases such as NonStop SQL/MP or NonStop SQL/MX. JDBC provides a call-level API for SQL-based database access.
Java HotSpot virtual machine	The Java virtual machine implementation designed to produce maximum program-execution speed for applications running in a server environment. This virtual machine features an adaptive compiler that dynamically optimizes the performance of running applications.
Java IDL	See Java Interface Development Language (Java IDL)
Java Interface Development Language (Java IDL)	The library that supports CORBA and Java interoperability. For more information, see the Sun Microsystems Java IDL documentation (http://java.sun.com/products/jdk/idl/index.html).
Java Naming and Directory Interface (JNDI)	A standard extension to the Java platform, which provides Java technology-enabled application programs with a unified interface to multiple naming and directory services.
Java Native Interface (JNI)	The C-language interface used by C functions called by Java classes. Includes an Invocation API that invokes a Java virtual machine from a C program.
Java Platform Standard Edition (Java SE 6)	The core Java technology platform, which provides a complete environment for applications development on desktops and servers and for deployment in embedded environments. For more information, see the Sun Microsystems JDK 6.0 Documentation.
Java runtime	See Java SE Runtime Environment.
Java SE Development Kit (JDK)	The development kit delivered with the Java SE platform. Contrast with Java SE Runtime Environment (JRE). See also, Java Platform Standard Edition 6.0 (Java SE).
Java SE Runtime Environment (JRE)	The Java virtual machine and the Core Packages. This is the standard Java environment that the java command invokes. Contrast with Java SE Development Kit (JDK). See also, Java Platform Standard Edition 6.0 (Java SE).
Java Transaction API (JTA)	An Oracle product that specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications. For more information, see the Sun Microsystems JTA document (http://java.sun.com/products/jta/index.html).
Java Transaction Service (JTS)	The transaction API , modeled on OMG's OTS. The NonStop Server for Java 6.0 includes an implementation of the its current interface.
Java virtual machine (JVM)	The process that loads, links, verifies, and interprets Java bytecode. The NonStop Server for Java 6.0 implements the Java HotSpot Server virtual machine.
Java Virtual Machine Tool Interface (JVM TI)	A programming interface used by development and monitoring tools. It is used to inspect the state and to control the execution of applications running in the Java VM, thereby defining the debugging services a VM provides.

JavaBeans A platform-neutral component architecture supported by Java, intended for developing or assembling software for multiple hardware and operating system environments. For more information, see the Sun Microsystems JavaBeans document (http://java.sun.com/javase/6/docs/technotes/guides/beans/index.html).

JavaBeans Development Kit (BDK) A set of tools for creating JavaBeans that is included with the NonStop Server for Java 6.0.

`javac` The Java compiler, which compiles Java source code into bytecode. Also, the command to run the Java compiler.

`javachk` The Java Checker, which determines whether a problem with the Java virtual machine is due to an incorrect TCP/IP configuration. Also, the command to run the Java checker.

`javadoc` The Java API documentation generator, which generates API documentation in HTML format from Java source code. Also, the command to run the Java API documentation generator.

`javah` The C header and Stub file generator, which generates C header files and C source files from a Java class, providing the connections that allow Java and C code to interact. Also, the command to run the C header and stub file generator.

`javap` The Java class file disassembler, which disassembles compiled Java files and prints a representation of the Java bytecode. Also, the command to run the Java class file disassembler.

JCK See Java Conformance Kit (JCK).

`jdb` The Java Debugger, which helps you find and fix errors in Java programs. Also, the command to run the Java Debugger `.jdb` uses the Java Debugger API.

JDBC See Java Database Connectivity (JDBC).

JDBC/MP See HPE JDBC Driver for SQL/MP (JDBC/MP).

JDBC/MX See HPE JDBC Driver for SQL/MX (JDBC/MX).

JDK See Java SE Development Kit (JDK).

JNDI See Java Naming and Directory Interface (JNDI).

JNI See Java Native Interface (JNI).

`jre` The Java runtime environment, which executes Java bytecode. See also Java SE Runtime Environment (JRE).

JTA See Java Transaction API (JTA).

JTS See Java Transaction Service (JTS).

`jts.Current` A JTS interface that lets you define transaction boundaries. The NonStop Server for Java 6.0 includes an implementation of `jts.Current`.

JVM See Java virtual machine (JVM).

JVM TI See Java Virtual Machine Tool Interface (JVM TI).

K-M

key

1. A value used to identify a record in a database, derived by applying a fixed function to the record. The key is often simply one of the fields (a column if the database is considered as a table with records being rows). Alternatively, the key can be obtained by applying a function to one or more of the fields.
2. A value that must be fed into the algorithm used to decode an encrypted message in order to reproduce the original plain text. Some encryption schemes use the same (secret) key to encrypt and decrypt a message, but public key encryption uses a private (secret) key and a public key that is known by all parties.

LAN See local area network (LAN).

LOB Short for Large OBject. Represents either `CLOB` or `BLOB` data.

local area network (LAN) A data communications network that is geographically limited (typically to a radius of 1 kilometer), allowing easy interconnection of terminals, microprocessors, and computers within adjacent buildings. Ethernet is an example of a LAN.

macro A sequence of commands that can contain dummy arguments. When the macro runs, actual parameters are substituted for the dummy arguments.

MXCI SQL/MX Conversational Interface.

N

native In the context of Java programming, something written in a language other than Java (such as C or C++) for a specific platform.

native method A non-Java routine (written in a language such as C or C++) that is called by a Java class.

`native2ascii` The Native-to-ASCII converter, which converts a file with native-encoded characters into one with Unicode-encoded characters. Also, the command to run the Native-to-ASCII converter.

node
1. An addressable device attached to a computer network.
2. A hypertext document.

NonStop Server for Java 5 The informal name of the NonStop Server for Java, based on the Java 2 Platform Standard Edition 5.0 products. This product is a Java environment that supports compact, concurrent, dynamic, and portable programs for the enterprise server.

NonStop Technical Library The browser-based interface to NonStop computing technical information. NonStop Technical Library replaces HPE Total Information Manager (TIM).

NSK See HPE NonStop operating system.

NSKCOM A program management tool for swap space.

O

Object Management Group (OMG) The standards body that defined CORBA.

Object Serialization An Oracle procedure that extends the core Java Input/Output classes with support for objects by supporting the following:

- The encoding of objects, and the objects reachable from them, into a stream of bytes.
- The complementary reconstruction of the object graph from the stream.

Object Serialization is used for lightweight persistence and for communication by means of sockets or RMI. The default encoding of objects protects private and transient data, and supports the evolution of the classes. A class can implement its own external encoding and is then solely responsible for the external format.

Object Transaction Service (OTS) The transaction service standard adopted by the OMG and used as the model for JTS.

ODBC See Open Database Connectivity (ODBC).

OLTP See online transaction processing (OLTP).

OMG See Object Management Group (OMG).

online transaction processing (OLTP) A method of processing transactions in which entered transactions are immediately applied to the database. The information in the databases is always current with the state of company and is readily available to all users through online screens and printed reports. The transactions are processed while the requester waits, as opposed to queued or batched transactions, which are processed at a later time. Online transaction processing can be used for many different kinds of business tasks, such as order processing, inventory control, accounting functions, and banking operations.

Open Database Connectivity (ODBC) The standard Microsoft product for accessing databases.

Open System Services (OSS) An environment available for interactive and programmatic use with the NonStop operating system. Processes that run in the OSS environment use the OSS API. Interactive users of the OSS environment use the OSS shell for their command interpreter. Compare to Guardian.

OSS See Open System Services (OSS).

OTS See Object Transaction Service (OTS).

P

package A collection of related classes; for example, JDBC.

Pathsend API The application program interface to a Pathway system that enables a Pathsend process to communicate with a server process.

Pathsend process A client (requester) process that uses the Pathsend interface to communicate with a server process. A Pathsend process can be either a standard requester, which initiates application requests, or a nested server, which is configured as a server class but acts as a requester by making requests to other servers. Also called a Pathsend requester.

Pathway A group of software tools for developing and monitoring OLTP programs that use the client / server model. Servers are grouped into server classes to perform the requested processing. On NonStop systems, this group of tools is packaged as two separate products: TS/MP and Pathway/TS.

Pathway CGI An extension to iTP Secure WebServer that provides CGI -like access to Pathway server classes. Extended in the NonStop Server for Java so that Java servlets can be invoked from a `ServletServerClass`, a special Pathway CGI server.

Pathway/TS A Hewlett Packard Enterprise product that provides tools for developing and interpreting screen programs to support OLTP programs in the Guardian environment on NonStop servers. Pathway/TS screen programs communicate with terminals and intelligent devices. Pathway/TS requires the services of the TS/MP product.

persistence

1. A property of a programming language where created objects and variables continue to exist and retain their values between runs of the program.
2. The capability of continuing in existence, such as a program running as a process.

portability The ability to transfer programs from one platform to another without reprogramming. A characteristic of open systems. Portability implies use of standard programming languages such as C.

Portable Operating System Interface X (POSIX) A family of interrelated interface standards defined by ANSI and IEEE. Each POSIX interface is separately defined in a numbered ANSI/IEEE standard or draft standard. The standards deal with issues of portability, interoperability, and uniformity of user interfaces.

POSIX See Portable Operating System Interface X (POSIX).

private key An encryption key that is not known to all parties.

protocol A set of formal rules for transmitting data, especially across a network. Low-level protocols define electrical and physical standards, bit-ordering, byte-ordering, and the transmission, error detection, and error correction of the bit stream. High-level protocols define data formatting, including the syntax of messages, the terminal-to-computer dialogue, character sets, sequencing of messages, and so on.

Pthread A POSIX thread.

public key An encryption key that is known to all parties.

pure Java Java that relies only on the Core Packages, meaning that it can run anywhere.

R

_RLD_LIB_PATH The location where the Java VM and other Java programs search for the TNS/E jdbcMx PIC file. Set _RLD_LIB_PATH explicitly or with the `_RLD_LIB_PATH` environment variable.

RDF See Remote Duplicate Database Facility (RDF).

Remote Duplicate Database Facility (RDF) The Hewlett Packard Enterprise software product that does the following:

- Assists in disaster recovery for OLTP production databases.
- Monitors database updates audited by the TMF subsystem on a primary system and applies those updates o a copy of the database on a remote system.

Remote Method Invocation (RMI) The Java package used for homogeneous distributed objects in an all-Java environment.

requester See client.

RMI See Remote Method Invocation (RMI).

`rmic` The Java RMI stub compiler, which generates stubs and skeletons for remote objects.

`rmicregistry` The Java Remote Object Registry, which starts a remote object registry on the specified port on the current host.

S

scalability The ability to increase the size and processing power of an online transaction processing system by adding processors and devices to a system, systems to a network, and so on, and to do so easily and transparently without bringing systems down. Sometimes called expandability.

Scalable TCP/IP (SIP) A NonStop Server for Java feature that transparently provides a way to give scalability and persistence to a network server written in Java.

serialization See Object Serialization.

serialized object An object that has undergone object serialization.

`serialver` The Serial Version Command, which returns the `serialVersionUID` of one or more classes. Also, the command to run the Serial Version Command.

server

1. An implementation of a system used as a stand-alone system or as a node in an Expand network.
2. The hardware component of a computer system designed to provide services in response to requests received from clients across a network. For example, NonStop system servers provide transaction processing, database access, and other services.
3. A process or program that provides services to a client. Servers are designed to receive request messages from clients; perform the desired operations, such as database inquiries or updates, security verifications, numerical calculations, or data routing to other computer systems; and return reply messages to the clients.

servlet A server -side Java program that any World Wide Web browser can access. It inherits scalability and persistence from the Pathway CGI server that manages it.

The Java class named servlets executes in server environments such as World Wide Web servers. The Servlet API is defined in a draft standard by Oracle. The `servlets` class is not in the Core Packages for the JDK.

set() method A method used to modify a data item. For example, the `SQLMPConnection.setAutoCommit()` method changes the transaction mode of the JDBC driver's connection to an SQL/MP or SQL/MX database. Compare to `get()` method.

shell The command interpreter used to pass commands to an operating system; the part of the operating system that is an interface to the outside world.

SIP See Scalable TCP/IP (SIP).

skeleton In RMI , the complement of the stub. Together, skeletons and stubs form the interface between the RMI services and the code that calls and implements remote objects.

SQL/MP See HPE NonStop SQL/MP.

SQL/MX See HPE NonStop SQL/MX.

SQLJ Also referred to SQLJ Part 0 the "Database Language SQL—Part 10: Object Language Bindings (SQL/OLB) part of the ANSI SQL-2002 standard that allows static SQL statements to be embedded directly in a Java program.

Standard Extension API An API outside the Core Packages for which Oracle has defined and published an API standard. Some of the Standard Extensions might migrate into the Core Packages. Examples of standard extensions are servlets and JTS.

stored procedure A procedure registered with NonStop SQL/MX and invoked by NonStop SQL/MX during execution of a CALL statement. Stored procedures are especially important for client/server database

systems because storing the procedure on the server side means that it is available to all clients. And when the procedure is modified, all clients automatically get the new version.

stored procedure in Java (SPJ) A stored procedure whose body is a static Java method.

stub
1. A dummy procedure used when linking a program with a runtime library. The stub need not contain any code. Its only purpose is to prevent "undefined label" errors at link time.
2. A local procedure in a remote procedure call (RPC). A client calls the stub to perform a task, not necessarily aware that the RPC is involved. The stub transmits parameters over the network to the server and returns results to the caller.

T

TACL See HPE Tandem Advanced Command Language (TACL).

TCP/IP See Transmission Control Protocol/Internet Protocol (TCP/IP).

Technical Documentation Hewlett Packard Enterprise's Technical documentation is found at http://www.hpe.com/support/hpesc

thread A task that is separately dispatched and that represents a sequential flow of control within a process.

`threads` The nonnative thread package that is shipped with Sun Microsystems Java SE 6.0.

`throw` Java keyword used to raise an exception.

`throws` Java keyword used to define the exceptions that a method can raise.

TMF See HPE NonStop Transaction Management Facility (TMF)

TNS/E The hardware platform based on the Intel Itanium architecture and the HPE NonStop operating system, and the software specific to that platform. All code is PIC (position independent code).

TNS/R The hardware platform based on the MIPS architecture and the HPE NonStop operating system, and the software specific to that platform. Code might be PIC (position independent code) or non-PIC.

transaction A user-defined action that a client program (usually running on a workstation) requests from a server.

Transaction Management Facility (TMF) A set of Hewlett Packard Enterprise software products for NonStop systems that assures database integrity by preventing incomplete updates to a database. It can continuously save the changes that are made to a database (in real time) and back out these changes when necessary. It can also take online "snapshot" backups of the database and restore the database from these backups.

Transmission Control Protocol/Internet Protocol (TCP/IP) One of the most widely available nonvendor-specific protocols , designed to support large, heterogeneous networks of systems.

trigger A trigger defines a set of actions that are executed automatically whenever a delete, insert, or update operation occurs on a specified base table.

U-Z

Unicode A character-coding scheme designed to be an extension of ASCII. By using 16 bits for each character (rather than ASCII's 7), Unicode can represent almost every character of every language and many symbols (such as "&") in an internationally standard way, eliminating the complexity of incompatible extended character sets and code pages. Unicode's first 128 codes correspond to those of standard ASCII.

uniform resource locator (URL) A draft standard for specifying an object on a network (such as a file, a newsgroup, or, with JDBC, a database). URLs are used extensively on the World Wide Web. HTML documents use them to specify the targets of hyperlinks.

URL See uniform resource locator (URL).

virtual machine (VM)	A self-contained operating environment that behaves as if it is a separate computer. See also Java virtual machine and Java Hotspot virtual machine.
VM	See virtual machine (VM).
World Wide Web (WWW)	An Internet client - server hypertext distributed information retrieval system that originated from the CERN High-Energy Physics laboratories in Geneva, Switzerland. On the WWW everything (documents, menus, indexes) is represented to the user as a hypertext object in HTML format. Hypertext links refer to other documents by their URLs. These can refer to local or remote resources accessible by FTP, Gopher, Telnet, or news, as well as those available by means of the HTTP protocol used to transfer hypertext documents. The client program (known as a browser) runs on the user's computer and provides two basic navigation operations: to follow a link or to send a query to a server.
wrapper	A shell script that sets up the proper execution environment and then executes the binary file that corresponds to the shell's name.
WWW	See World Wide Web (WWW).