JDBC Type 4 Programmer's Reference Guide

Version 2.1.0

Table of Contents

1. About This Document 4
    1.1. Intended Audience 4
    1.2. New and Changed Information 4
    1.3. Notation Conventions 4
    1.4. Comments Encouraged 8
2. Introduction 9
    2.1. Type 4 Driver API Package 9
    2.2. Installation 9
3. Accessing Trafodion SQL Databases 10
    3.1. Data Sources 10
        3.1.1. JDBC Data Source (client-side) 10
    3.2. Security 10
    3.3. Connection by Using the DataSource Interface 11
        3.3.1. Overview of Tasks to Deploy DataSource Objects 11
        3.3.2. DataSource Object Properties 12
        3.3.3. Programmatically Creating an Instance of the DataSource Class 12
        3.3.4. Programmatically Registering the DataSource Object 13
        3.3.5. Retrieving a DataSource Instance by Using JNDI and Connecting to the Data Source 13
        3.3.6. Specifying the Properties File that Configures the Data Source 14
    3.4. Connection by Using the DriverManager Class 14
        3.4.1. Loading and Registering the Driver 15
        3.4.2. Establishing the Connection 16
        3.4.3. Guidelines for Connections Using the Driver Manager 16
    3.5. Connection Pooling 18
    3.6. Statement Pooling 19
        3.6.1. Guidelines for Statement Pooling 19
        3.6.2. Troubleshooting Statement Pooling 19
    3.7. Thread-Safe Database Access 20
    3.8. "Update . . . Where Current of" Operations 20
    3.9. INFOSTATS Command for Obtaining Query Costs 21
        3.9.1. Use of the INFOSTATS Command 22
    3.10. Internationalization Support 24
        3.10.1. When String Literals Are Used in Applications 24
        3.10.2. Controlling String Literal Conversion by Using the Character-Set Properties 24
        3.10.3. Localizing Error Messages and Status Messages 27
4. Type 4 Driver Properties 29
    4.1. Summary of Type 4 Driver Properties 29
        4.1.1. Client-Side Properties 29
        4.1.2. Server-Side Properties 31
    4.2. How to Specify JDBC Type 4 Properties 31
        4.2.1. Where to Set Properties 32

4.2.2. Creating and Using a Properties File . . . 32
4.2.3. Setting Properties in the Command Line . . . 33
4.2.4. Precedence of Property Specifications . . . 33
5. Type 4 Driver Property Descriptions . . . 34
5.1. catalog Property . . . 34
5.2. connectionTimeout Property . . . 35
5.3. fetchBufferSize Property . . . 36
5.4. initialPoolSize Property . . . 37
5.5. ISO88591 Property . . . 38
5.6. KANJI Property . . . 39
5.7. KSC5601 Property . . . 40
5.8. language Property . . . 41
5.9. loginTimeout Property . . . 42
5.10. maxIdleTime Property . . . 43
5.11. maxPoolSize Property . . . 44
5.12. maxStatements Property . . . 45
5.13. minPoolSize Property . . . 46
5.14. networkTimeout Property . . . 47
5.15. password Property . . . 48
5.16. properties Property . . . 48
5.17. reserveDataLocators Property . . . 49
5.18. roundingMode Property . . . 50
5.19. schema Property . . . 51
5.20. T4LogFile Property . . . 52
5.21. T4LogLevel Property . . . 54
5.21.1. T4LogLevel Considerations . . . 55
5.22. translationVerification Property . . . 56
5.23. url Property . . . 57
5.23.1. url Property Considerations . . . 57
5.24. user Property . . . 58
6. Type 4 Driver Compliance . . . 59
6.1. Compliance Overview . . . 59
6.2. Unsupported Features . . . 59
6.3. Deviations . . . 61
6.4. Trafodion Extensions . . . 63
6.4.1. Internationalization of Messages . . . 63
6.4.2. Additional DatabaseMetaData APIs . . . 63
6.5. Conformance of DatabaseMetaData Methods' Handling of Null Parameters . . . 63
6.6. Type 4 Driver Conformance to SQL Data Types . . . 65
6.6.1. JDBC Data Types . . . 65
6.7. Floating-Point Support . . . 66
6.8. SQLJ Support . . . 66
6.9. JDBC 3.0 Features Not Supported by the Type 4 Driver . . . 67
6.10. Restrictions . . . 67

7. Tracing and Logging Facilities . . . 68
7.1. Standard JDBC Tracing and Logging Facility . . . 68
7.2. The Type 4 Driver Logging Facility . . . 69
7.2.1. Controlling Type 4 Driver Logging Output . . . 69
7.2.2. Message Format . . . 70
7.2.3. Examples of Logging Output . . . 71
8. Messages . . . 72
8.1. About the Message Format . . . 72
8.2. Getting Help . . . 72
8.3. Type 4 Driver Error Messages . . . 72
8.3.1. 01032 08S01 . . . 72
8.3.2. 01056 25000 . . . 72
8.3.3. 01118 S1008 . . . 73
8.3.4. 08001 HY000 . . . 73
8.3.5. 08004 HY000 . . . 73
8.3.6. 29001 HYC00 . . . 74
8.3.7. 29002 08003 . . . 74
8.3.8. 29003 HY000 . . . 74
8.3.9. 29004 HY024 . . . 75
8.3.10. 29005 HY024 . . . 75
8.3.11. 29006 HY000 . . . 75
8.3.12. 29007 07009 . . . 76
8.3.13. 29008 24000 . . . 76
8.3.14. 29009 HY109 . . . 76
8.3.15. 29010 07009 . . . 77
8.3.16. 29011 07009 . . . 77
8.3.17. 29012 07006 . . . 77
8.3.18. 29013 HY024 . . . 78
8.3.19. 29015 HY024 . . . 78
8.3.20. 29017 HY004 . . . 78
8.3.21. 29018 22018 . . . 79
8.3.22. 29019 07002 . . . 79
8.3.23. 29020 07009 . . . 79
8.3.24. 29021 HY004 . . . 80
8.3.25. 29022 HY010 . . . 80
8.3.26. 29026 HY000 . . . 80
8.3.27. 29027 HY011 . . . 81
8.3.28. 29029 HY011 . . . 81
8.3.29. 29031 HY000 . . . 81
8.3.30. 29032 23000 . . . 82
8.3.31. 29033 23000 . . . 82
8.3.32. 29035 HY000 . . . 82
8.3.33. 29036 HY000 . . . 83
8.3.34. 29037 HY106 . . . 83

8.3.35. 29038 HY107 83
8.3.36. 29039 HY092 84
8.3.37. 29040 HY000 84
8.3.38. 29041 HY000 84
8.3.39. 29042 HY000 85
8.3.40. 29043 HY000 85
8.3.41. 29044 HY000 85
8.3.42. 29045 01S07 86
8.3.43. 29046 22003 86
8.3.44. 29047 HY000 86
8.3.45. 29048 HY009 87
8.3.46. 29049 25000 87
8.3.47. 29050 HY107 87
8.3.48. 29051 01S02 88
8.3.49. 29053 HY000 88
8.3.50. 29054 HY000 88
8.3.51. 29056 HY000 89
8.3.52. 29057 HY000 89
8.3.53. 29058 HY000 89
8.3.54. 29059 HY000 90
8.3.55. 29060 HY000 90
8.3.56. 29061 HY00 90
8.3.57. 29063 HY00 91
8.3.58. 29067 07009 91
8.3.59. 29068 07009 91
8.3.60. 29069 HY000 92
8.3.61. 29100 HY000 92
8.3.62. 29101 HY000 92
8.3.63. 29102 HY000 93
8.3.64. 29103 HY000 93
8.3.65. 29104 HY000 93
8.3.66. 29105 HY000 94
8.3.67. 29106 HY000 94
8.3.68. 29107 HY000 94
8.3.69. 29108 HY000 95
8.3.70. 29109 HY000 95
8.3.71. 29110 HY000 95
8.3.72. 29111 HY000 96
8.3.73. 29112 HY000 96
8.3.74. 29113 HY000 96
8.3.75. 29114 HY000 97
8.3.76. 29115 HY000 97
8.3.77. 29116 HY000 97
8.3.78. 29117 HY000 98

8.3.79. 29118 HY000 98
8.3.80. 29119 HY000 98
8.3.81. 29120 HY000 99
8.3.82. 29121 HY000 99
8.3.83. 29122 HY000 99
8.3.84. 29123 HY000 100
8.3.85. 29124 HY000 100
8.3.86. 29125 HY000 100
8.3.87. 29126 HY000 101
8.3.88. 29127 HY000 101
8.3.89. 29128 HY000 101
8.3.90. 29129 HY000 102
8.3.91. 29130 HY000 102
8.3.92. 29131 HY000 102
8.3.93. 29132 HY000 103
8.3.94. 29133 HY000 103
8.3.95. 29134 HY000 103
8.3.96. 29135 HY000 104
8.3.97. 29136 HY000 104
8.3.98. 29137 HY000 104
8.3.99. 29138 HY000 105
8.3.100. 29139 HY000 105
8.3.101. 29140 HY000 105
8.3.102. 29141 HY000 106
8.3.103. 29142 HY000 106
8.3.104. 29143 HY000 106
8.3.105. 29144 HY000 107
8.3.106. 29145 HY000 107
8.3.107. 29146 HY000 107
8.3.108. 29147 HY000 108
8.3.109. 29148 HY000 108
8.3.110. 29149 HY000 108
8.3.111. 29150 HY000 109
8.3.112. 29151 HY000 109
8.3.113. 29152 HY000 109
8.3.114. 29153 HY000 110
8.3.115. 29154 HY000 110
8.3.116. 29155 HY000 110
8.3.117. 29156 HY000 111
8.3.118. 29157 HY000 111
8.3.119. 29158 HY000 111
8.3.120. 29159 HY000 112
8.3.121. 29160 HY000 112
8.3.122. 29161 S1000 112

8.3.123. 29162 S1000 . . . 113
8.3.124. 29163 08001 . . . 113
8.3.125. 29164 08001 . . . 113
8.3.126. 29165 HY000 . . . 114
8.3.127. 29166 HY000 . . . 114
8.3.128. 29167 HY000 . . . 114
8.3.129. 29168 HY000 . . . 115
8.3.130. 29169 HY000 . . . 115
8.3.131. 29170 HY000 . . . 115
8.3.132. 29172 HY000 . . . 116
8.3.133. 29173 HY000 . . . 116
8.3.134. 29174 HY000 . . . 116
8.3.135. 29175 HY000 . . . 117
8.3.136. 29177 HY000 . . . 117
8.3.137. 29178 HY000 . . . 117
8.3.138. 29182 HY000 . . . 118
8.3.139. S1000 HY000 . . . 118
9. Avoiding Driver-Server Version Mismatch . . . 119
9.1. Compatible Versions . . . 119
9.2. Considerations for Mixed-Version JDBC Clients Connecting to Trafodion Platforms . . . 119
9.3. Version Mismatch Error Message . . . 120

License Statement

Disclaimer: *Apache Trafodion is an effort undergoing incubation at the Apache Software Foundation (ASF), sponsored by the Apache Incubator PMC. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF. <<<*

Acknowledgements

Revision History

Version	Date
2.0.1	July 7, 2016
2.0.0	June 6, 2016
1.3.0	January, 2016

Chapter 1. About This Document

This document describes how to use the Trafodion JDBC Type 4 Driver (subsequently called the Type 4 driver). This driver provides Java applications running on a foreign platform with JDBC access to Trafodion.

1.1. Intended Audience

This Trafodion JDBC Type 4 Driver Programmer's Reference Guide is for experienced Java programmers who want to access Trafodion SQL databases.

This document assumes you are already familiar with the Java documentation, which is located at http://docs.oracle.com/en/java/.

1.2. New and Changed Information

This is a new manual.

1.3. Notation Conventions

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.

  ```
  SELECT
  ```

- lowercase letters

  Lowercase letters, regardless of font, indicate variable items that you supply. Items not enclosed in brackets are required.

  ```
  file-name
  ```

- [ ] Brackets

  Brackets enclose optional syntax items.

```
DATETIME [start-field TO] end-field
```

  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:

```
DROP SCHEMA schema [CASCADE]
DROP SCHEMA schema [ CASCADE | RESTRICT ]
```

- { } Braces

  Braces enclose required syntax items.

```
FROM { grantee [, grantee ] ... }
```

  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:

```
INTERVAL { start-field TO end-field }
{ single-field }
INTERVAL { start-field TO end-field | single-field }
```

- | Vertical Line

  A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces.

```
{expression | NULL}
```

- … 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.

```
ATTRIBUTE[S] attribute [, attribute] ...
{, sql-expression } ...
```

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

  For example:

```
expression-n ...
```

- Punctuation

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

```
DAY (datetime-expression)
@script-file
```

  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:

```
"{" module-name [, module-name] ... "}"
```

- Item Spacing

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

```
DAY (datetime-expression) DAY(datetime-expression)
```

  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:

```
myfile.sh
```

- 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.

```
match-value [NOT] LIKE _pattern
   [ESCAPE esc-char-expression]
```

1.4. Comments Encouraged

We encourage your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to user@trafodion.incubator.apache.org.

Include the document title and any comment, error found, or suggestion for improvement you have concerning this document.

Chapter 2. Introduction

This document describes how to use the Trafodion JDBC Type 4 Driver. This driver provides Java applications running on a foreign platform with JDBC access to Trafodion SQL databases on the Trafodion.

Supported Java Releases: The Type 4 driver requires Java enabled platforms that support JDK 1.4.1 or higher.

2.1. Type 4 Driver API Package

The Type 4 driver package, `org.trafodion.jdbc.t4`, is shipped with the driver software. For class and method descriptions, see the *Trafodion JDBC Type 4 Driver API Reference*.

The Trafodion JDBC Type 4 Driver (hereafter, Type 4 driver) implements JDBC technology that conforms to the standard JDBC 3.0 Data Access API.

To obtain detailed information on the standard JDBC API, download the JDBC API documentation: http://docs.oracle.com/en/java/.

2.2. Installation

Refer to the Trafodion Client Installation Guide.

Chapter 3. Accessing Trafodion SQL Databases

3.1. Data Sources

The term **data source** logically refers to a database or other data storage entity. A JDBC (client) data source is physically a Java object that contains properties such as the URL of the physical database, the catalog to use when connecting to this database, and the schema to use when connecting to this database. The JDBC data source also contains methods for obtaining a JDBC connection to the underlying database.

3.1.1. 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 `org.trafodion.jdbc.t4.HPT4DataSource` and `org.trafodion.jdbc.t4.HPT4ConnectionPoolDataSource`. (These classes are defined by the JDBC 3.0 specification.)

Typically, a user or system administrator uses a tool to create a data source, and then registers 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 a DCS (Data Connectivity Services) data source.

3.2. Security

Clients connect to the Trafodion platform with a valid user name and password, using standard JDBC 3.0 APIs. An application can make multiple connections using different user IDs, and creating different Connection objects.

The Type 4 driver provides for user name and password authentication. The password is encrypted.

3.3. Connection by Using the DataSource Interface

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.

When an application requests a connection by using the `getConnection` method in the `DataSource`, then 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.

3.3.1. Overview of Tasks to Deploy 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 administrator using a GUI tool as part of the installation of the data source. If you are using an installed data source, then see Programmatically Creating an Instance of the DataSource Class.

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

For more information about using data sources, see Connecting with DataSource Objects in the JDBC™ Database Access: Table of Contents documentation or other information available in the field.

3.3.2. 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 (the primary IP address or host name of the database), the database schema and catalog names, the location of the database server, the name of the database, and so forth.

For details about Type 4 driver properties that you can use with the `DataSource` object, see Type 4 Driver Properties.

3.3.3. 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 `HPT4DataSource` object or the `HPT4ConnectionPoolDataSource` object.

Example

```
HPT4DataSource temp = new HPT4DataSource() ;
temp.setCatalog( "TRAFODION" ) ;
```

In the following example, the code fragment illustrates the methods that a `DataSource` object `ds` needs to include if the object supports the `serverDataSource` property `ds.setServerDataSource( "my_server_datasource" )`. In this example, the code shows setting properties for the `HPT4DataSource` object to use the Type 4 driver to access a Trafodion database:

```
HPT4DataSource ds = new HPT4DataSource() ;

ds.setUrl( "jdbc:t4jdbc://<primary IP addr or host name>:2300/" );
ds.setCatalog( "TRAFODION" ) ;
ds.setSchema( "myschema" ) ;
ds.setUser( "gunnar" ) ;
ds.setPassword( "my_userpassword" ) ;

// 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" ) ;
```

This technique essentially builds a properties file. For more information, see Creating and Using a Properties File.

3.3.4. Programmatically Registering the DataSource Object

In the following example, the code shows how to register, programmatically, the `HPT4DataSource` object `ds` that was created using the preceding code with JNDI.

```
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 ) ;
```

3.3.5. Retrieving a DataSource Instance by Using JNDI and Connecting to the Data Source

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

The steps that JDBC application does to connect to and use the data source associated with the database are listed below 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 - server.

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

4. Create the connection using the data source.

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

5. Do work with the connection. The following statements are just a simple example.

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

3.3.6. 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 for more information about creating this file.

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

```
java -Dhpt4jdbc.properties=<path of properties file on disk>
```

3.4. Connection by Using the DriverManager Class

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.

`org.trafodion.jdbc.t4.T4Driver` is the Type 4 driver class that implements the `java.sql.Driver` interface.

3.4.1. Loading and Registering the Driver

Before connecting 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=org.trafodion.jdbc.t4.T4Driver
```

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

```
Class.forName("org.trafodion.jdbc.t4.T4Driver")
```

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

```
jdbc.drivers=org.trafodion.jdbc.t4.T4Driver
```

3.4.2. 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:t4jdbc://<ip addr or host name>:23400/[:][property=value[;property2=value2]...]
```

Parameter	Usage
`<ip addr or host name>`	The primary IP address or host name for the Trafodion database.
`23400`	The port number for the Trafodion SQL database.
`property = value` and `property2=value2`	Specifies 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.

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

```
Class.forName( "org.trafodion.jdbc.t4.T4Driver" ) ; //loads the driver

String url = "jdbc:hpt4jdbc://<database primary IP address>:23400/"

Connection con = DriverManager.getConnection( url, "userID", "Passwd" ) ;
```

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

3.4.3. Guidelines for Connections Using the Driver Manager

- The Type 4 driver defines a set of properties that you can use to configure the driver. For detailed information about these properties, see Type 4 Driver Properties.
- Java applications can specify the properties in these 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:t4jdbc://<ip addr or host name>:23400/:property=value
     ```

     `<ip addr or host name>` is the primary IP address or host name for the Trafodion database.

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:

   ```
   -Dt4jdbc.properties=<path of properties file on disk>
   ```

   For example, `-Dt4jdbc.properties=C:\temp\t4props`

   For information about the properties file, see Creating and Using a Properties File.

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:

   ```
   -Dt4jdbc.property_name=<property value>
   ```

   For example, `-Dt4jdbc.maxStatements=1024`

3.5. 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 these Type 4 driver properties:
  - `maxPoolSize` under maxpoolsize Property
  - `minPoolSize` under minPoolSize Property
  - `initialPoolSize` under initialPoolSize Property
  - `maxStatements` under maxStatements Property
- 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 these combination of properties:

```
url
catalog
schema
username
password
```

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

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.

3.6. Statement Pooling

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

3.6.1. Guidelines for Statement Pooling

- To enable statement pooling, set the `maxStatements` property to an integer value greater than 0 and enable connection pooling. For more information, see initialPoolSize Property and Connection Pooling.
- Enabling statement pooling for your JDBC applications might dramatically improve the performance.
- Explicitly close a prepared statement 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 these methods:
  - `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.

3.6.2. Troubleshooting Statement Pooling

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

- The Type 4 driver looks for a matching `PreparedStatement` object in the statement pool and reuses the `PreparedStatement`. The matching criteria include the SQL string, catalog, current schema, current transaction isolation, and result set holdability.

  If the Type 4 driver finds the matching `PreparedStatement` object, then 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 compilation.

  If this condition is not true, then reuse of a `PreparedStatement` object might result in unexpected behavior.

- Avoid recompiling to yield performance improvements from statement pooling. The SQL 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.
- When a query is recompiled, then the SQL 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 does not cache `Statement` objects.

3.7. Thread-Safe Database Access

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

- `HPT4DataSource.getConnection()` is implemented as a synchronized method to ensure thread safety in getting a connection.
- Once a connection is made, the `Connection` object is instance-specific.
- If multiple statements are run on different threads in a single connection, then statement objects are serialized to prevent data corruption.

3.8. "Update . . . Where Current of" Operations

The fetch size on a `ResultSet` must be 1 when performing an `update . . . where current of` *cursor* SQL statement.

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.

The following is an example of setting a result set's 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
                     ) ;
```

3.9. INFOSTATS Command for Obtaining Query Costs

The INFOSTATS command reports the roll-up costs of a particular query. INFOSTATS is a pass-through command that collects statistics for a prepared statement. Statistics are returned to the JDBC application as a result set as soon as the prepare is finished. The result set has these columns:

Column	Description
`Query ID (SQL_CHAR)`	The unique identifier for the query.
`CPUTime (SQL_DOUBLE)`	An estimate of the number of seconds of processor time it might take to execute the instructions for this query. A value of 1.0 is 1 second.
`IOTime (SQL_DOUBLE)`	An estimate of the number of seconds of I/O time (seeks plus data transfer) to perform the I/O for this query.
`MsgTime (SQL_DOUBLE)`	An estimate of the number of seconds it takes for the messaging for this query. The estimate includes the time for the number of local and remote messages and the amount of data sent.
`IdleTime (SQL_DOUBLE)`	An estimate of the maximum number of seconds to wait for an event to happen for this query. The estimate includes the amount of time to open a table or start an ESP process.
`TotalTime (SQL_DOUBLE)`	Estimated cost associated to execute the query.
`Cardinality (SQL_DOUBLE)`	Estimated number of rows that will be returned.

3.9.1. Use of the INFOSTATS Command

The INFOSTATS command can only be used with PreparedStatement objects. The syntax is:

```
INFOSTATS cursor_name
```

where `cursor_name` is the name of the prepared statement. If the cursor name is case-sensitive, then enclose it in single quotes.

To get the cursor name, use the `getStatementLabel()` method that is defined for the Trafodion JDBC Type 4 driver with class:

```
org.trafodion.jdbc.t4.T4PreparedStatement: public String
getStatementLabel() ;
```

Considerations

- You can use INFOSTATS in these methods only:

  ```
  java.sql.Statement.executeQuery(String sql)
  java.sql.Statement.execute(String sql)
  ```

- `setCursorName` is not supported with INFOSTATS.
- If you invoke INFOSTATS incorrectly, the Type 4 driver issues this error:

  ```
  Message: INFOSTATS command can only be executed
           by calling execute(String sql) method.
           Sqlstate HY000
           Sqlcode 29180
  ```

Example of INFOSTATS

```
Statement s = conn.createStatement( ) ;

HPT4PreparedStatement p =
   (HPT4PreparedStatement)conn.prepareStatement(
      "SELECT * FROM t WHERE i = ?" ) ;

boolean results = s.execute( "INFOSTATS " + p.getStatementLabel() ) ;

if ( results )
{
   ResultSet rs = s.getResultSet( ) ;

   while ( rs.next( ) )
   {
      //process data
   }
}
```

Sample Output

```
QueryID: MXID00100112821201636991234819l_16_SQL_CUR_9829657
CPUTime: 0.09975778464794362
IOTime: 0.10584000146627659
MsgTime: 0.09800000134418951
IdleTime: 0.09800000134418951
TotalTime: 0.10584000146627659
Cardinality: 100.0
```

3.10. Internationalization Support

3.10.1. When String Literals Are Used in Applications

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

1. When the driver processes an SQL statement. For example,

```
Statement stmt = connection.getStatement() ;

stmt.execute( "SELECT * FROM table1 WHERE col1 = 'abcd'" ) ;
```

2. When 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 to an array of bytes for processing by the Trafodion, the Type 4 driver uses the column type in the database.

3.10.2. 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 character-set formats to and from the Java string Unicode (`UnicodeBigUnmarked`) encoding.

The Type 4 driver provides character-set mapping properties through key values as shown in the following table.

Key	Default Value
ISO88591	ISO88591_1
KANJI	SJIS
KSC5601	EUC_KR

A description of these character sets appears in table below, which summarizes the character sets supported by Trafodion.

Trafodion Character Set	Corresponding Java Encoding Set[1]	Description
ISO88591	ISO88591_1	Single-character, 8-bit character-data type ISO88591 supports English and other Western European languages.

[1] Canonical Name for `java.io` and `java.lang` API.

For detailed information, see ISO88591 Property.

Using the Character-Set Properties

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

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

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 -Dhpt4jdbc.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.

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 What Happens on 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, then 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, then 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 translationVerification Property.

3.10.3. 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 form:

```
T4Messages_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 `T4Messages_en.properties`.

Localized-Message String Format

A localized message file contains strings in the form:

```
message=message_text
```

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} Error detail: {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.

Procedure to Create a Localized-Message File

1. Extract the `T4Messages_en.properties file`, which is in the `jdbcT4-*.jar file`.

   Example

   From a UNIX prompt, use the jar Java tool: `jar -x T4Messages_en.properties < jdbcT4-*.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 the Localized-Messages File.
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.

Chapter 4. Type 4 Driver Properties

4.1. Summary of Type 4 Driver Properties

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

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

4.1.1. Client-Side Properties

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.

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`	Sets 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.)

Internationalization Properties

Property Name	Description	Default Value
`ISO88591`	Sets character-set mapping that corresponds to the ISO88591 character set.	ISO88591_1
`KANJI`	Sets character-set mapping that corresponds to the KANJI character set.	SJIS (which is shift-JIS, Japanese)
`KSC5601`	Sets character-set mapping that corresponds to the KSC5601 character set.	ECU_KR (which is KS C 5601, ECU encoding, Korean)
`language`	Sets the language used for 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

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/t4jdbc%u.log`
`T4LogLevel`	Sets the logging levels that control logging output for the Type 4 driver.	OFF

Miscellaneous Client-Side Properties

Property Name	Description	Default Value
`description`	Specifies the registered source name.	None.
`fetchBufferSize`	Provides the benefits of bulk fetch when rows are fetched from a `ResultSet` object.	4 kilobytes
`properties`	Specifies the location of the properties file that contains keyword-value pairs that specify property values for configuring the Type 4 driver.	None.
`roundingMode`	Specifies the rounding behavior of the Type 4 driver.	ROUND_DOWN

4.1.2. Server-Side Properties

The 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.

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. Must be "TRAFODION" in the current release.
`connectionTimeout`	Sets the number of seconds a connection can be idle before the connection is physically closed by DCS.	-1 (Use the ConnTimeout value set on the server data source.)
`password`	Sets the password value for passing to the database. Can also change the password.	Empty string.
`schema`	Sets the database schema that accesses SQL objects referenced in SQL statements if the SQL objects are not fully qualified.	None.
`url`	Sets the URL value for the database. Can be set only on the `DriverManager` object.	None.
`user`	Sets the user value for the database.	None.

4.2. How to Specify JDBC Type 4 Properties

The Type 4 JDBC driver properties configure the driver. These properties can be specified in a data source, a connection URL (the primary IP address or host name on the database), a properties file, or in the java command line.

Java properties have the form:

```
key=value
```

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

4.2.1. Where to Set 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:
  1. Using the option `-Dproperty_name=property_value` in the command line.
  2. Using the `java.util.Properties` parameter in the `getConnection()` method of the `DriverManager` class.

4.2.2. 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:

```
-Dt4jdbc.properties=<path of the properties file on disk>`
```

Example

```
-Dt4jdbc.properties=C:\temp\t4props\myprops.properties
```

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
```

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:t4jdbc://<primary IP addr or host name of database>:23400/
user=database_username
password=mypassword
description=<a string>
catalog=TRAFODION
schema=myschema
maxPoolSize=20
minPoolSize=5
maxStatements=20
loginTimeout=15
initialPoolSize=10
connectionTimeout=10
T4LogLevel=OFF
T4LogFile=/mylogdirectory/mylogfile
```

4.2.3. 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: `t4jdbc`.

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

Example

The maxStatements property becomes:

```
-Dt4jdbc.maxStatements=10
```

4.2.4. 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 `t4jdbc.properties` property.
4. Set the `-Dt4jdbc.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.

Chapter 5. Type 4 Driver Property Descriptions

The properties are listed in alphabetic order with their descriptions. For the properties summarized in categories, see Summary of Type 4 Driver Properties.

5.1. catalog Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

Default: none
```

Example

Specifying the catalog TRAFODION:

```
catalog=TRAFODION
```

5.2. connectionTimeout Property

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

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: int

Units: seconds

Default: -1 (Use the ConnTimeout value set on the server-side 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 Trafodion data source, if allowed by the connectivity settings.
- A negative value is treated as -1.

Example

Consider the following 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
```

5.3. fetchBufferSize Property

The `fetchBufferSize` property provides the benefits of bulk fetch.

This property sets the value in kilobytes (KB) of the size of the fetch buffer that is used when rows are fetched from a `ResultSet` object after a successful `executeQuery()` operation on a statement.

Set this property on a `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: short

Default size: 4

Range: 4 through 32767
```

- Zero and negative values are treated as default values.
- The Type 4 driver guarantees that the number of rows internally fetched will be no less than the minimum of the row size (set using the `setFetchSize` method) and the number of rows that will fit in the memory specified by the `setFetchSize` (set using the property).

Example

```
fetchBufferSize=32
```

5.4. initialPoolSize Property

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, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

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, then 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.

Example

```
initialPoolSize=10
```

5.5. ISO88591 Property

The `ISO88591` character-set mapping property corresponds to the SQL 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 Support.

Set this property on a `DataSource` object or `DriverManager` object. This property is ignored for connections made through the `ConnectionPoolDataSource` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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 Java documentation.

For more information, see Internationalization Support.

5.6. KANJI Property

The `KANJI` character-set mapping property corresponds to the SQL 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 big endian.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

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

Example

```
java -Dt4jdbc.KANJI=SJIS
```

For more information, see Internationalization Support.

5.7. KSC5601 Property

The `KSC5601` character-set mapping property corresponds to the SQL KSC5601 character set, which is a double-byte character set.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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 Java documentation.

Example

```
java -Dt4jdbc.KSC5601=ECU_KR
```

For more information, see Internationalization Support.

5.8. language Property

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

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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 Java documentation.

Example

To set the language to shift-JIS, Japanese:

```
language=SJIS
```

5.9. loginTimeout Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: int

Units: seconds

Default: 60

Range: 0 to 2147483647
```

If set to 0 (zero), no login timeout is specified.

5.10. maxIdleTime Property

The `maxIdleTime` property determines the number of seconds that a physical connection should 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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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.

Example

To set the maximum idle time to 5 minutes (300 seconds):

```
java -Dhpt4jdbc.maxIdleTime=300
```

5.11. maxPoolSize Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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.

5.12. maxStatements Property

The `maxStatements` property sets the total number of `PreparedStatement` objects that the connection pool should cache. This total includes both free objects and objects in use.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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).

To improve performance, we recommend that you enable statement pooling for your JDBC applications because this pooling can dramatically help the performance of many applications.

Statement pooling can be in effect only if connection pooling is enabled.

Example

To specify statement pooling, type:

```
maxStatements=10
```

5.13. minPoolSize Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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.

Example

Use the following specification to set the `minPoolSize` value to 1, which ensures that one connection is always retained:

```
minPoolSize=1
```

5.14. networkTimeout Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

Be careful when using this property. A network timeout causes the socket connection between the Type 4 driver and the connectivity 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, the transaction manager 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
```

5.15. password Property

The `password` property sets the password value for passing to the DCS server. By using this property, you can also change the password. The password is encrypted when it is passed to the server.

The format for specifying the password is:

```
password=old [, new, new ]
```

- `old` is the current password
- `new` is the new password. Passwords must be 6 to 8 characters long and cannot contain double quotes (").

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

Default: empty string
```

Example

```
password=eye0weU$
```

5.16. properties Property

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.

5.17. reserveDataLocators Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
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 executed. If the application inserts a large number of LOB items, then 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, then a smaller value is better. If a large value is used, then holes (unused data-locator numbers) could occur in the LOB table. These holes represent unused space.

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.

To change this value for a JDBC application, specify this property from the command line.

Example

The following command reserves 150 data locators for program class `myProgramClass`.

```
java -Dhpt4jdbc.reserveDataLocators=150 myProgramClass
```

5.18. roundingMode Property

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()`, then the value returned is 1234.12, which is truncated as specified by the default rounding mode, `ROUND_DOWN`.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

Default: ROUND_DOWN
```

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.
- If the application sets erroneous values for the `roundingMode` property, no error is thrown by the Type 4 driver. The driver uses `ROUND_DOWN` value instead.
- To have the application get the `DataTruncation` exception when data is truncated, set the `roundingMode` property to `ROUND_UNNECESSARY`.

5.19. schema Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

Default: none
```

Example

```
schema=sales
```

5.20. T4LogFile Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String
```

Default file name is defined by the following pattern:

```
%h/t4jdbc%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, then 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.

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.

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 documentation.

5.21. T4LogLevel Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

Default: OFF
```

Logging Levels

Level	Description
`OFF`	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.

Example

To enable tracing, use the `hpt4jdbc.T4LogLevel` property specified in the command line:

```
-Dt4jdbc.T4LogLevel=FINE
```

5.21.1. T4LogLevel Considerations

- If a security manager is defined by your application using an AppServer, then `LoggingPermission` must be 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.

5.22. translationVerification Property

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. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

The value can be TRUE or FALSE.

```
Data type: String

Default: FALSE
```

Value	Scenario	What Happens
FALSE	The driver is unable to translate all or part of an SQL statement, then 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.
TRUE	The driver cannot translation all or part of an SQL statement or parameter.	The driver throws an SQLException with this 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, then the process can use significantly more system resources. For better performance, set this property to FALSE.

For more information, see Internationalization Support.

5.23. url Property

The `url` property sets the URL value for the database. This property is used in the `DriverManager` object. The format to specify the URL is:

```
jdbc:t4jdbc//<primary IP addr or hostname of database>:23400/[:]
[ property=value [ ; property2=value ] ... ]
```

where `<primary IP_addr or hostname of database>:23400>` specifies the location of the database.

```
Data type: String

Default: none
```

Example

```
url=jdbc:t4jdbc://mynode.mycompanynetwork.net:23400/
```

5.23.1. url Property Considerations

- If the url parameter is not specified and `DriverManager.getConnection()` is called, then the Type 4 driver throws an SQLException.
- If you use a literal IPV4 or IPV6 address in a URL, note these 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 database is `23400`.

5.24. user Property

The `user` property sets the role value for the connectivity service. The role name passed must have adequate access permissions for SQL data accessed through the connectivity service.

Set this property on a `DataSource` object, `ConnectionPoolDataSource` object, or `DriverManager` object. For information about how to set properties, see How to Specify JDBC Type 4 Properties.

```
Data type: String

Default: empty string
```

Example

```
user=System_rolename
```

Chapter 6. Type 4 Driver Compliance

6.1. Compliance Overview

The Type 4 driver conforms where applicable to the 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 Trafodion 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 Trafodion SQL that are not supported by the Trafodion JDBC Type 4 driver, other unsupported features, and restrictions.

6.2. Unsupported Features

These methods in the java.sql package throw an SQLException with the message `Unsupported feature - <method-name>`:

Method	Comments

`CallableStatement.getArray(int parameterIndex)`
`CallableStatement.getArray(String parameterName)`
`CallableStatement.getBlob(int parameterIndex)`
`CallableStatement.getBlob(String parameterName)`
`CallableStatement.getClob(int parameterIndex)`
`CallableStatement.getClob(String parameterName)`
`CallableStatement.getObject(int parameterIndex, Map map)`
`CallableStatement.getObject(String parameterName, Map map)`
`CallableStatement.getRef(int parameterIndex)`
`CallableStatement.getRef(String parameterName)`
`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.

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

Method	Comments

`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.

`Statement.getQueryTimeout()` `Statement.setQueryTimeout()`	The particular `Statement` methods are not supported.

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(String parameterIndex, 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:

Method	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 Trafodion.

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

Method	Comments
`javax.sql.XAConnection` `javax.sql.XADataSource`	Distributed Transactions, as described in the JDBC 3.0 API specification, are not yet implemented.

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

6.3. 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.

The `java.sql.DatabaseMetaData.getVersionColumns()` method mimics the `java.sql.DatabaseMetaData.getBestRowIdentifier()` method because Trafodion SQL 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 Trafodion SQL does not support the column type for these types:
	
	`SCOPE_CATALOG,`
	`SCOPE_SCHEMA,`
	`SCOPE_TABLE,`
	and `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 Trafodion SQL does not support the column type for these types:
	
	`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 Trafodion SQL does not support the base type.

Method	Comments
`java.sql.DatabaseMetaData.getVersionColumns( )`	Mimics the `DatabaseMetaData.getBestRowIdentifier()` method because Trafodion SQL 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 SQL Warning is issued if an application requests that type. The result set is changed to a scroll-insensitive type.
`java.sql.ResultSet.setFetchDirection(. . .)`	The fetch direction attribute is ignored.
`java.sql.Statement.cancel()`	In some instances, drops the connection to the server instead of just canceling the query. You must then reconnect to the server. Note that the connection is dropped if `cancel()` is issued for a statement that is being processed. Otherwise the connection is maintained.
`java.sql.Statement.setEscapeProcessing(. . .)`	Because Trafodion SQL parses the escape syntax, disabling escape processing has no effect.
`java.sql.Statement.setFetchDirection(. . .)`	The fetch direction attribute is ignored.

6.4. Trafodion Extensions

The Trafodion extensions to the JDBC standard implemented in the Type 4 driver are as follows.

6.4.1. 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 Messages and Status Messages.

6.4.2. Additional DatabaseMetaData APIs

APIs added to the `T4DatabaseMetaData` class provide these capabilities:

- Get a description of a table's synonyms.

```
public java.sql.ResultSet getSynonymInfo(String catalog, String schema, String
table) throws SQLException
```

6.5. 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 % wild card.
- 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 % wild card retrieves data for all schemas of the specified catalog

6.6. Type 4 Driver Conformance to SQL Data Types

6.6.1. JDBC Data Types

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

JDBC Data Type	Support by JDBC Driver for Trafodion SQL	Trafodion SQL Data Type
`Types.Array`	No	Not applicable.
`Types.BIGINT`	Yes	`LARGEINT`
`Types.BINARY`	Data type is mapped by Trafodion SQL. Data type varies from that used for table creation.	`CHAR(n)`[1]
`Types.BIT`	Data type is mapped by Trafodion SQL. Data type varies from that used for table creation.	`CHAR(1)`
`Types.CHAR`	Yes	`CHAR(n)`
`Types.DATE`	Yes	`DATE`
`Types.DECIMAL`	Yes	`DECIMAL(p,s)`
`Types.DISTINCT`	No	Not applicable.
`Types.DOUBLE`	Yes	`DOUBLE PRECISION`
`Types.FLOAT`	Yes	`FLOAT(p)`
`Types.INTEGER`	Yes	`INTEGER`
`Types.JAVA_OBJECT`	No	Not applicable.
`Types.LONGVARBINARY`	Data type is mapped by Trafodion SQL. Data type varies from that used for table creation.	`VARCHAR(n)`[1]
`Types.LONGVARCHAR`	Yes. Maximum length is 4018.	`VARCHAR[(n)]`
`Types.NULL`	No	Not applicable.
`Types.NUMERIC`	Yes	`NUMERIC(p,s)`
`Types.REAL`	Yes	`FLOAT(p)`
`Types.REF`	No	Not applicable.
`Types.SMALLINT`	Yes	`SMALLINT`
`Types.STRUCT`	No	Not applicable.
`Types.TIME`	Yes	`TIME`
`Types.TIMESTAMP`	Yes	`TIMESTAMP`
`Types.TINYINT`	Data type is mapped by Trafodion SQL. Data type varies from that used for table creation.	`SMALLINT`
`Types.VARBINARY`	Data type is mapped by Trafodion SQL. Data type varies from that used for table creation.	`VARCHAR(n)`[1]
`Types.VARCHAR`	Yes	`VARCHAR(n)`
`Types.BOOLEAN`	Data type is mapped by Trafodion SQL. Data type varies from that used for table creation.	`CHAR(1)`
`Types.DATALINK`	No	Not applicable.

[1] Because of mapping provided by Trafodion, 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`:

```
INTERVAL YEAR(p)
INTERVAL YEAR(p) TO MONTH
INTERVAL MONTH(p)
INTERVAL DAY(p)
INTERVAL DAY(p) TO HOUR
INTERVAL DAY(p) TO MINUTE
INTERVAL DAY(p) TO SECOND
INTERVAL HOUR(p)
INTERVAL HOUR(p) TO MINUTE
INTERVAL HOUR(p) TO SECOND
INTERVAL MINUTE(p)
INTERVAL MINUTE(p) TO SECOND
INTERVAL SECOND(p)
```

6.7. 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.

6.8. SQLJ Support

The Type 4 driver supports non-customized SQLJ applications, but does not support customized SQLJ applications.

6.9. JDBC 3.0 Features Not Supported by the Type 4 Driver

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

- Multiple result sets returned by batch statements.
- Database savepoint support. (Not provided in Trafodion SQL)
- 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 Type 4 driver and DCS.
- Security context (user name and password) implicit propagation from AppServer to the Type 4 driver.
- IPV6 protocol stack. (IPV6 addressing is emulated over IPV4 on the Trafodion platform - server side)
- Distributed transactions.

6.10. Restrictions

- The Type 4 driver supports only database features that are supported by Trafodion SQL and SPJ. Therefore, the Type 4 driver is not fully compliant with JDBC 3.0 specifications.
- The Type 4 driver depends on DCS (Data Connectivity Service) for all server side manageability related features.

Chapter 7. Tracing and Logging Facilities

The Type 4 driver provides two tracing and logging facilities:

- Standard JDBC tracing and logging functionality as defined by the JDBC standard
- Type 4 driver logging facility

Server-side tracing (logging) is enabled by configuring DCS.

7.1. 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.

- 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 so on. 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 DriverSource interface API.

7.2. The Type 4 Driver Logging Facility

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

7.2.1. 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 information about using this property, see T4LogLevel Property.
- `T4LogFile`: Specifies the file to which the driver is to write logging information. For information about using this property, see T4LogFile Property.

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

Example

These properties file entries set the logging level to SEVERE and specify a log file name:

```
T4LogLevel= SEVERE

T4LogFile=c:/T4logfile1.log
```

7.2.2. 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]
```

Identifier	Provides
`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 Java VM.
`connection-id`	If applicable, a unique ID for the connection associated with the message.

`server-id`	If applicable, information about the connectivity server associated with the message. The *server-id* is of the form:
	
	`TCP:node-name.server-name/port-number:ODBC`
	
	where
	
	`node-name` is the name of the Trafodion database node.
	`server-name` is the name of the Trafodion platform.
	`port-number` is the port to which the server is connected.
	
	Example
	
	`TCP:\banshee-tcp.$Z0133/46003:ODBC`

`dialogue-id`	If applicable, the `dialogue-id` used for the DCS 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.

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 prepended with zeros (0) to allow for readable formatting.

7.2.3. Examples of Logging Output

- Output where `T4LogLevel` is set to SEVERE:

```
00000036 ~ Dec 8, 2006 10:05:55 AM PST ~ 10 ~ 4508606 ~ null
    ~ null ~ T4Messages.createSQLException("en_US",
    "socket_write_error", "null") ~
```

- 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 )
```

Chapter 8. Messages

8.1. About the Message Format

Messages are listed in numerical SQLCODE order. Descriptions include:

```
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. ]
```

8.2. Getting Help

Some messages have no recovery information. Please contact user@trafodion.incubator.apache.org for assistance.

8.3. Type 4 Driver Error Messages

8.3.1. 01032 08S01

```
01032 08S01 Communication link failure. The server timed out or disappeared.
```

Cause: The connection timed out.

Effect: Operation fails.

Recovery: Reconnect. Set the connection timeout to an appropriate value.

8.3.2. 01056 25000

```
01056 25000 Invalid transaction state.
```

Cause: Transaction state is incorrect.

Effect: Operation fails.

Recovery: Retry.

8.3.3. 01118 S1008

```
01118 S1008 Operation canceled.
```

Cause: The operation was canceled.

Effect: Operation fails.

Recovery: Retry operation.

8.3.4. 08001 HY000

```
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: Contact user@trafodion.incubator.apache.org to check logs for server errors and to analyze accompanying errors and warnings.

8.3.5. 08004 HY000

```
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.

8.3.6. 29001 HYC00

```
HYC00 Unsupported feature - {0}
```

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

Effect: An unsupported exception is thrown, and a NULL `resultSet` is returned.

Recovery: Remove the feature functionality from the program.

8.3.7. 29002 08003

```
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.

8.3.8. 29003 HY000

```
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.

8.3.9. 29004 HY024

```
29004 HY024 Invalid transaction isolation value.
```

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

Effect: `HPT4Connection.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`.

8.3.10. 29005 HY024

```
29005 HY024 Invalid ResultSet type
```

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

Effect: The SQL Statement call with the `resultSetType` parameter fails.

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

8.3.11. 29006 HY000

```
29006 HY000 Invalid Result Set concurrency
```

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

Effect: The `HPT4Statement` call with `resultSetConcurrency` fails.

Recovery: Valid resultSetConcurrency values are: CONCUR_READ_ONLY and CONCUR_UPDATABLE.

8.3.12. 29007 07009

```
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.

8.3.13. 29008 24000

```
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.

8.3.14. 29009 HY109

```
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.

8.3.15. 29010 07009

```
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.

8.3.16. 29011 07009

```
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.

8.3.17. 29012 07006

```
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.

8.3.18. 29013 HY024

```
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.

8.3.19. 29015 HY024

```
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`.

8.3.20. 29017 HY004

```
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.

8.3.21. 29018 22018

```
29018 2018 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.

8.3.22. 29019 07002

```
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.

8.3.23. 29020 07009

```
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.

8.3.24. 29021 HY004

```
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`, and `byte[]`, `Blob`, and `Clob`.

8.3.25. 29022 HY010

```
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.

8.3.26. 29026 HY000

```
29026 HY000 Transaction can't be committed or rolled back when AutoCommitmode 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.

8.3.27. 29027 HY011

```
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.

8.3.28. 29029 HY011

```
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.

8.3.29. 29031 HY000

```
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.

8.3.30. 29032 23000

```
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.

8.3.31. 29033 23000

```
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.

8.3.32. 29035 HY000

```
29035 HY000IO Exception occurred {0}

message_text
```

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.

8.3.33. 29036 HY000

```
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.

8.3.34. 29037 HY106

```
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_ONLYResultSet` object type cursors can move forward only. `TYPE_SCROLL_SENSITIVE` and `TYPE_SCROLL_INSENSITIVE` types are scrollable.

8.3.35. 29038 HY107

```
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).

8.3.36. 29039 HY092

```
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`.

8.3.37. 29040 HY000

```
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.

8.3.38. 29041 HY000

```
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.

8.3.39. 29042 HY000

```
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.

8.3.40. 29043 HY000

```
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.

8.3.41. 29044 HY000

```
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.

8.3.42. 29045 01S07

```
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.

8.3.43. 29046 22003

```
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.

8.3.44. 29047 HY000

```
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.

8.3.45. 29048 HY009

```
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.

8.3.46. 29049 25000

```
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.

8.3.47. 29050 HY107

```
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. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.48. 29051 01S02

```
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.

8.3.49. 29053 HY000

```
29053 HY000 SQL SELECT statement is invalid in executeUpdate() methodCause.
```

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.

8.3.50. 29054 HY000

```
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.

8.3.51. 29056 HY000

```
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.

8.3.52. 29057 HY000

```
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.

8.3.53. 29058 HY000

```
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.

8.3.54. 29059 HY000

```
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 `hpt4jdbc.blobTableName`, or the property is set to an invalid value.

Effect: The application cannot access BLOB columns.

Recovery: Set the `hpt4jdbc.blobTableName` property to a valid LOB table name. The LOB table name is of format `catalog.schema.lobTableName`.

8.3.55. 29060 HY000

```
29060 HY000 'hpt4jdbc.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 `propertyhpt4jdbc.clobTableName` property, or the property is set to null value or set to an invalid value.

Effect: The application cannot access CLOB columns.

Recovery: Set the `hpt4jdbc.clobTableName` property to a valid LOB table name. The LOB table name is of format `catalog.schema.lobTableName`.

8.3.56. 29061 HY00

```
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.

8.3.57. 29063 HY00

```
29063 HY00 Transaction error {0} - {1} while obtaining start data locator.
```

Cause: A transaction error occurred when the Type 4 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.

8.3.58. 29067 07009

```
2067 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.

8.3.59. 29068 07009

```
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.

8.3.60. 29069 HY000

```
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.

8.3.61. 29100 HY000

```
29100 HY000 An internal error occurred.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.62. 29101 HY000

```
29101 HY000 Contact your service provider.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org (the service provider) and report the entire message.

8.3.63. 29102 HY000

```
29101 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.

8.3.64. 29103 HY000

```
29103 HY000 Address is null.
```

Cause: The address was empty.

Effect: Operation fails.

Recovery: Refer to url Property for the valid address format.

8.3.65. 29104 HY000

```
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.

8.3.66. 29105 HY000

```
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.

8.3.67. 29106 HY000

```
29016 HY000 Expected address format: jdbc:subprotocol::subname.
```

Cause: Not applicable.

Effect: Not applicable.

Recovery: This is an informational message. Refer to url Property for the valid address format.

8.3.68. 29107 HY000

```
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.

8.3.69. 29108 HY000

```
29108 HY000 Expecting \\<machine-name><process-name>/<port-number>.
```

Cause: The DCS address format was invalid.

Effect: Operation fails.

Recovery: The address returned by the Trafodion platform was not in the expected format. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.70. 29109 HY000

```
29109 HY000 //<{IP Address|Machine Name}[:port]/database name>
```

Cause: Informational message.

Effect: Not applicable.

Recovery: Not applicable.

8.3.71. 29110 HY000

```
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.

8.3.72. 29111 HY000

```
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 may not be properly qualified or there my exist a security restriction. 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.

8.3.73. 29112 HY000

```
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 url Property for the valid address format.

8.3.74. 29113 HY000

```
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.

8.3.75. 29114 HY000

```
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.

8.3.76. 29115 HY000

```
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.

8.3.77. 29116 HY000

```
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.

8.3.78. 29117 HY000

```
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.

8.3.79. 29118 HY000

```
29118 HY000 A write to a bad map pointer occurred.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.80. 29119 HY000

```
29119 HY000 A write to a bad par pointer occurred.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.81. 29120 HY000

```
29120 HY000 An association server connect message error occurred.
```

Cause: Unable to connect to the DCS association server.

Effect: Operation fails.

Recovery: Evaluate the returned value from the `getCause` method on the `Exception` to determine the appropriate recovery action.

8.3.82. 29121 HY000

```
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.

8.3.83. 29122 HY000

```
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.

8.3.84. 29123 HY000

```
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.

8.3.85. 29124 HY000

```
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.

8.3.86. 29125 HY000

```
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.

8.3.87. 29126 HY000

```
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.

8.3.88. 29127 HY000

```
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.

8.3.89. 29128 HY000

```
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.

8.3.90. 29129 HY000

```
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.

8.3.91. 29130 HY000

```
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.

8.3.92. 29131 HY000

```
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.

8.3.93. 29132 HY000

```
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.

8.3.94. 29133 HY000

```
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.

8.3.95. 29134 HY000

```
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.

8.3.96. 29135 HY000

```
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.

8.3.97. 29136 HY000

```
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 and contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze accompanying errors and warnings.

8.3.98. 29137 HY000

```
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.

8.3.99. 29138 HY000

```
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.

8.3.100. 29139 HY000

```
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.

8.3.101. 29140 HY000

```
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.

8.3.102. 29141 HY000

```
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.

8.3.103. 29142 HY000

```
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.

8.3.104. 29143 HY000

```
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.

8.3.105. 29144 HY000

```
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.

8.3.106. 29145 HY000

```
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.

8.3.107. 29146 HY000

```
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.

8.3.108. 29147 HY000

```
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.

8.3.109. 29148 HY000

```
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.

8.3.110. 29149 HY000

```
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.

8.3.111. 29150 HY000

```
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.

8.3.112. 29151 HY000

```
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.

8.3.113. 29152 HY000

```
29152 HY000 No more ports available to start ODBC servers.
```

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors. Evaluate the returned value from the `getCause` method on the `Exception` to determine the appropriate recovery action.

8.3.114. 29153 HY000

```
29153 HY000 Invalid authorization specification.
```

Cause: Incorrect user name and/or password.

Effect: Operation fails.

Recovery: Retry with correct user name and/or password.

8.3.115. 29154 HY000

```
29154 HY000 Timeout expired.
```

Cause: Unable to perform this operation.

Effect: Operation fails.

Recovery: Retry and/or change the timeout value for the operation.

8.3.116. 29155 HY000

```
29155 HY000 Unknown message type.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze errors and warnings.

8.3.117. 29156 HY000

```
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. Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze accompanying errors and warnings.

8.3.118. 29157 HY000

```
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. Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze accompanying errors and warnings.

8.3.119. 29158 HY000

```
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: Evaluate any error or error detail information accompanying the message. Install compatible versions of the driver and HP connectivity server.

8.3.120. 29159 HY000

```
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 HP connectivity server.

8.3.121. 29160 HY000

```
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. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.122. 29161 S1000

```
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. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.123. 29162 S1000

```
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: Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze accompanying errors and warnings.

8.3.124. 29163 08001

```
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 server-side data source with more servers.

8.3.125. 29164 08001

```
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.

8.3.126. 29165 HY000

```
29165 HY000 Unknown connect reply error: <error>.
```

Cause: A message returned by the server.

Effect: Operation fails.

Recovery: Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze accompanying errors and warnings.

8.3.127. 29166 HY000

```
29166 HY000 This method is not implemented.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.128. 29167 HY000

```
29167 HY000 Internal error. An internal index failed consistency check.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.129. 29168 HY000

```
29168 HY000 Unknown reply message error: <error> error detail: <error_detail>.
```

Cause: Server returned an error.

Effect: Operation fails.

Recovery: Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors and to analyze accompanying errors and warnings.

8.3.130. 29169 HY000

```
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. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.131. 29170 HY000

```
29170 HY000 Invalid parameter value.
```

Cause: Internal error.

Effect: Operation fails.

Recovery: None. Contact user@trafodion.incubator.apache.org and report the entire message.

8.3.132. 29172 HY000

```
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.

8.3.133. 29173 HY000

```
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.

8.3.134. 29174 HY000

```
29174 HY000 Autocommit is on and updateRow was called on the ResultSetobject.
```

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.

8.3.135. 29175 HY000

```
29175 HY000 Unknown Error {0}.
```

Cause: An unknown error occurred during connection {0}.

Effect: The connection fails.

Recovery: Retry the connection.

8.3.136. 29177 HY000

```
29177 HY000 Data cannot be null.
```

Cause: Attempted to get column value data in String format, but passed a null input value.

Effect: The operation fails.

Recovery: Contact user@trafodion.incubator.apache.org to check logs for server (Trafodion platform) errors regarding the DCS server.

8.3.137. 29178 HY000

```
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.

8.3.138. 29182 HY000

```
29182 HY000 General warning. Connected to the default data source:
TDM_Default_DataSource
```

Cause: The user application specified a data source that does not exist on the server side, the Trafodion platform.

Effect: The connection uses the Trafodion platform default data source `TDM_Default_DataSource`.

Recovery: Ignore the warning or contact your Trafodion database administrator to add the server-side data source that the application specified.

8.3.139. S1000 HY000

```
S1000 HY000 A TIP transaction error <error> has been detected. Check the
server event log on Node <segment> for Transaction Error details.
```

Cause: A message was returned by the server.

Effect: Operation fails.

Recovery: Contact user@trafodion.incubator.apache.org to check for errors in the server event log on the reported segment.

Chapter 9. Avoiding Driver-Server Version Mismatch

The Trafodion JDBC type 4 driver described in this document can connect only with an version-equivalent platform (server). It cannot connect with an earlier version platform.

To make a connection with the Trafodion platform, JDBC clients, through the driver, connect with the Trafodion database connectivity service (DCS) on the Trafodion platform. In some situations, Trafodion JDBC clients need to make connections to older-version platforms (servers) from the same client boxes. To make a connection, the driver version must be compatible with the Trafodion platform version.

The DCS release version and Trafodion platform release version are always the same.

9.1. Compatible Versions

Ensure that you install the driver version that is compatible with the Trafodion platform version.

Driver version	Compatible versions of the Trafodion platform
Trafodion Release 2.0 driver	All versions up to, but not including, Trafodion Release 2.2
Trafodion Release 2.1 driver	All versions up to, but not including, Trafodion Release 2.2
Trafodion Release 2.2 driver	Trafodion Release 2.2 and later versions

If a compatible version is not installed, you can obtain the software to download from the Trafodion download site.

9.2. Considerations for Mixed-Version JDBC Clients Connecting to Trafodion Platforms

On the client platform, you can install multiple versions of the Trafodion JDBC type 4 driver to connect to Trafodion platforms of different platform versions.

- Assuming you have installed the Release 2.2 Trafodion JDBC type 4 driver on your workstation and set up the client environment, the 2.2 driver's classes are set your java CLASSPATH.
- To connect to a Release 2.1 or 2.0 server (Trafodion platform) from the same client machine, you must load the 2.1 driver by making sure that it is in your java CLASSPATH.
- Connecting to both a 2.1 and 2.2 server from the same application at the same time is not possible.
- A given application must use either the 2.2 driver or the 2.1 driver when launched. The only way to switch is to reload the application when pointing to a new CLASSPATH that contains a different driver.

9.3. Version Mismatch Error Message

If an Trafodion JDBC client attempts to connect to an invalid DCS version, the driver returns the error:

```
SQLCODE: 29162
SQLSTATE S1000

Error text:

Unexpected programming exception has been found: <errortext>. Check
the server event log on node <logfile_location> for details.
```

- <errortext> is the error text from the server.
- <logfile_location> is the location of the log file.