Novell®

SQL *Connector*

JDBC Programmer's Guide

Printing Date: **October 1, 1999**

About This Manual

Purpose of this Manual

This manual describes the programming and use of SQL *C*–JDBC, which is an interface that enables you to use Java JDBC classes to create applications that connect with SQL *Connector* Data Sources. These applications can run on NetWare or Windows platforms, since Java is supported on both platforms. The SQL *Connector* Data Source is used to maintain and manage a reference list of physical databases and tables. These database table references are stored in the Data Source, and used by SQL *C*–JDBC to connect applications to these databases.

This manual provides descriptions and examples that are, as nearly as possible, generic for all databases supported by SQL *Connector*. Unless stated as an exception, SQL statements and procedures developed with the material in this document will be portable across all supported databases, regardless of the specific SQL requirements of the database.

Intended Audience

This document is intended for programmers who will be creating and maintaining applications which use SQL *C*–JDBC to connect to physical databases. It provides details on creating JDBC URL's, summarizes JDBC API usage, and provides information on how the JDBC functions and SQL grammar map to SQL *C*–JDBC features.

This document assumes that programmers are familiar with the JDBC Application Programming Interface (API). For information about the JDBC API, you can download information using http://java.sun.com/products/jdbc/index.html.

Structure of this Manual

This manual consists of chapters which describes how to configure and use SQL *C*–JDBC.

Associated Documents

The SQL *Connector* document set contains these manuals:

- SQL *Connector Overview*
- SQL *Connector Installation Guide*
- SQL *Connector Administration Guide*
- SQL *Connector SQL Grammar Manual*
- SQL *Connector ODBC Programmer's Guide*
- SQL *Connector JDBC Programmer's Guide*

Operating System Conventions

When there are differences in commands, examples, or syntax between operating systems, the following abbreviations are used:

Abbreviation	Meaning
NetWare	the Novell NetWare operating system
Windows	the Microsoft Windows 95/98/NT operating systems

Table of Contents

1

Overview

1.1 Introduction

SQL *Connector* is a Data Request Broker for database access. It provides the capability of using standard Structured Query Language (SQL) to access data in tables in different databases. The tables are cataloged in an SQL *Connector* Data Source, which can then be accessed by client applications using the Java Database Connectivity (JDBC) interface.

The SQL *Connector* Data Source is created and maintained by the Data Source Administrator and documented in the *Administration Guide*.

1.2 Architecture

Java applications can access the SQL *C*–DRB (Data Request Broker) from any system that supports the Java environment. The client application uses Java SQL *C*–JDBC classes to connect to SQL *C*–DRB running on a network server. SQL *C*–DRB uses an SQL *Connector* Data Source to connect to Oracle on the network server or to remote ODBC databases using the SQL *Connector* ODBC Data Driver.

1.2.1 JDBC Driver Client Configuration

A Java client application uses the following SQL *C*–JDBC classes and layers to connect to the SQL *Connector* network server and the SQL *Connector* Data Driver:

- Programming Layer

```
java.sql.*     prototype JDBC API (JDK version 1.1)
sqli.sql.*     SQL C–JDBC implementation of JDBC API
```

- System Layer

```
vortex.*       SQL C–JDBC socket classes
java.net.*     Java socket services
classes.zip    Java runtime environment (with socket services)
```

1.2.2 Data Request Broker Configuration

The system runtime environment uses TCP/IP network sockets to communicate with the server. The server executes a program which listens for incoming requests on a specified TCP/IP socket.

- System Layer

```
network listener     module vtxnetd.nlm, vtx16.nlm
```

- Connection Layer

```
SQLC host          module sqlcmon.nlm, sqlc.nlm
```

```
Data Source        local and remote databases
```

The SQL *Connector* network listener runs on the Data Broker system (typically from the time the Data Broker system is booted). The listener responds to requests from a specified TCP/IP socket. When it receives a network request from the client, it creates a client thread (if one is not already created), which then changes the network request into a database request and calls the SQL *Connector* Data Broker engine. The Data Broker engine uses the Data Source to access physical databases or local files.

1.2.3 Database Drivers Configuration

The SQL *Connector* Data Drivers are used to connect the Data Source to local and remote databases. Remote connectivity is discussed in the *Administration Guide*.

1.3 JDBC Interface

JDBC (Java Database Connectivity) is an industry-standard API (application programming interface) for connecting client applications and data servers. For more information about the JDBC API, see http://java.sun.com/products/jdbc/index.html.

The JDBC API documentation specifies the names and arguments of functions which define the interface between the Java client application and the data source. However, within the documentation, there are levels of conformance to the JDBC standard, and there are sections of the standard which allow vendor-defined features (for example, datatypes and error messages). The purpose of this manual is to discuss the implementation and programming of SQL *C*–JDBC as it relates to the JDBC standard.

For a discussion of the SQL grammar supported by SQL *Connector*, see the *SQL Grammar Manual*.

Installation of SQL *C*–JDBC is discussed in the *Installation Guide*, and creation and maintenance of a Data Source is discussed in the *Administration Guide*. The remainder of this document assumes that SQL *C*–JDBC has been successfully installed and a Data Source is available for use.

1.4 JDBC Classpath

1.4.1 Windows

The Windows `CLASSPATH` system variable must be updated to include the path to the SQL *C*–JDBC Java classes. For Windows 95/98, the file `C:\AUTOEXEC.BAT` must be edited. For example, if the SQL *C*–JDBC classes have been installed on the directory `C:\SQLC`, then the following command can be used to update the CLASSPATH system variable:

```
SET CLASSPATH = C:\SQLC\SQLCJDBC.JAR;%CLASSPATH%
```

For Windows NT, the `CLASSPATH` environment variable can be set using Control Panel > System > Environment.

1.4.2 NetWare

The NetWare CLASSPATH system variable must be updated to include the path to the SQL *C*–JDBC Java classes. For Netware, the CLASSPATH is defined in the file SYS:ETC\JAVA.CFG. The SQLCJDBC.JAR should be copied to the directory SYS:JAVA\CLASSES, then the JAVA.CFG file should be edited. A sample JAVA.CFG with an added line is shown below:

```
JAVA_HOME=SYS:\JAVA
OSA_HOME=SYS:\JAVA
MGMT_HOME=SYS:\PUBLIC\MGMT
CLASSPATH=sys:\java\lib\classes.zip;sys:\java\classes;.
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\swing.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\jgl3.1.0.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\help.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\servertop.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\jndi.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\njcl.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\vbjorb.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\vbjapp.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\vbjtools.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\lib\ucs.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\
CLASSPATH=$CLASSPATH;$OSA_HOME\beans\NWDir.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\beans\NWSess.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\beans\NWPrtQAdm.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\beans\NWSockets.jar
CLASSPATH=$CLASSPATH;$OSA_HOME\classes\SQLCJDBC.JAR  <-- added line
```

After the file is edited, the NetWare Java Virtual Machine must be restarted. At the NetWare console, type:

```
java -exit
java
```

The CLASSPATH can be examined using the following command:

```
envset classpath
```

The output will be similar to:

```
classpath=sys:\java\lib\classes.zip;sys:\java\classes;.;SYS:\JAVA\lib\
swing.jar;SYS:\JAVA\lib\jgl3.1.0.jar;SYS:\JAVA\lib\help.jar;SYS:\JAVA\lib
\servertop.jar;SYS:\JAVA\lib\jndi.jar;SYS:\JAVA\lib\njcl.jar;SYS:\JAVA\
lib\vbjorb.jar;SYS:\JAVA\lib\vbjapp.jar;SYS:\JAVA\lib\vbjtools.jar;SYS:\
JAVA\lib\ucs.jar;SYS:\JAVA\;SYS:\JAVA\beans\NWDir.jar;SYS:\JAVA\beans\
NWSess.jar;SYS:\JAVA\beans\NWPrtQAdm.jar;SYS:\JAVA\beans\NWSockets.jar;
SYS:\JAVA\classes\SQLCJDBC.JAR
```

The output confirms the location of the jar file.

2

Data Source Setup and Connection

2.1 Introduction

This chapter discusses creating and configuring an SQL *Connector* JDBC database connection on a client system. An SQL *Connector* (SQL *C*) JDBC connection is a JDBC connection to an SQL *Connector* Data Request Broker (DRB) and a Data Source on a network server. An SQL *C*–JDBC connection requires the name of the Data Source, the name of the server that is running the Data Request Broker and the network protocol used to access the server. Once the SQL *C*–JDBC driver is loaded and an SQL *C*–JDBC connection is created, a JDBC application can connect to an SQL *Connector* Data Source using the standard JDBC API.

2.2 JDBC Driver Loading

The SQL *C*–JDBC installation procedure installs the SQL *C*–JDBC driver and related software. The directory which contains the SQL *C*–JDBC jar file (`sqlcjdbc.jar`) must be included in the `CLASSPATH` system variable.

The name of the SQL *C*–JDBC driver is `sqli.sql.sqliDriver`. To load the SQL *C*–JDBC driver in a Java program, use the following (or similar) syntax:

```
import java.sql.*
. . .
Driver d = (Driver)Class.forName("sqli.sql.sqliDriver").newInstance();
```

2.3 JDBC Driver Connection

After the SQL *C*–JDBC driver is loaded, an SQL *Connector* connection can be completed by using the following (or similar) Java syntax:

```
String url = jdbc:sqli:<Net URL, see discussion below>;
String uid = <username>;
String pwd = <password>;
. . .
Connection c = DriverManager.getConnection(url,uid,pwd);
```

Uniform Resource Locator (URL) Connection String

The URL is a string for the network connection, which is in the following format (symbol "[" indicates option, symbol "|" indicates choice):

```
jdbc:sqli://[user@|:password@|user:password@]host:port/datasource
        [;environs=switches]
```

username - optional

The name of the user for the network connection, followed by "@" if the password is omitted. If the username is not included in the URL, it must be supplied as the second argument of `DriverManager.getConnection(url,uid,pwd)`, or supplied in the connection properties list for `DriverManager.getConnection(url,prop)`.

password - optional

The password of the user for the network connection, preceded by ":" and followed by "@". If the password is not included in the URL, it must be supplied as the third argument of `DriverManager.getConnection(url,uid,pwd)`, or supplied in the driver connection properties list when `DriverManager.getConnection(url,prop)` is used.

host – required

The name of the server or internet address (format nnn.nnn.nnn.nnn) which is running SQL *C*–DRB. The name must be in the system hosts file.

Windows host file:
```
c:\winnt\system32\drivers\etc\hosts
```

port – required

A port number used by TCP/IP socket services, preceded by a ":". This number must match a number used when the SQL *Connector* listener is started on the server.

datasource

The name of the Data Source (no spaces).

Sample Data Source name:
```
demo
```

Note: A sample Data Source can be created using data files included in the software distribution directory. See the *Administration Guide* for details.

environs

SQL *Connector* switch values or environment variables. For a full description, see the *SQL Grammar Manual*. The format is: `name1=value1,name2=value2,...`

Example

An example of a complete URL is shown below:

```
jdbc:sqli://user:pass@server1:1958/demo; ...
      environs=SS_TRACE=yes,SMARTTRACE=sql.trc
```

Example Notes:

1. The Data Source is named `demo`.
2. TCP/IP port 1958 is used on node `server1`
3. There are two switches used in this example:
   a. SS_TRACE = yes

   This switch turns on database tracing. All activity with local and remote databases will be written to a trace file. This trace file is useful for debugging and optimizing database applications.

   b. SMARTTRACE = `sql.trc`

This switch names the trace file. If the name is not set, a default name will be used. The default name is s<year-day-number><hour-minute>.trc, for example, s0411634.trc, and the default directory is the location of the loaded programs.

Other Data Source switch variables are discussed in the *SQL Grammar Manual*.

2.4 Testing a JDBC Data Source Connection

2.4.1 Sample Programs

The NetWare PUBLIC directory contains a Samples subdirectory that has two programs for testing an SQL *C*–JDBC Data Source connection using two and three tiers. One program (jdbcping) verifies the installation by connecting from a client system (first tier) to a Data Source on a network server (second tier) and reporting success or failure. The other program (jdbcjoin) connects from a client (first tier) to a Data Source on a network server (second tier) and retrieves joined data using a remote ODBC Data Driver on a database server (third tier). Both programs are supplied in source form (java code) and compiled classes (class files). The source code is listed in Appendices D and E.

The following programs can be copied from the Netware PUBLIC directory to a Windows *client* system (i.e., the system that will run the sample programs). The file locations are:

```
SYS:PUBLIC\SQLC\samples\windows\jdbc\jdbcjoin.java
SYS:PUBLIC\SQLC\samples\windows\jdbc\jdbcjoin.class
SYS:PUBLIC\SQLC\samples\windows\jdbc\jdbcping.java
SYS:PUBLIC\SQLC\samples\windows\jdbc\jdbcping.class
```

2.4.2 Sample Data

A sample database with sample tables is also supplied. The sample data source is a Microsoft Access file that contains tables for departments (dept), jobs (job), employees (emp) and salaries (sal). This sample database must be copied to a Windows remote system that has the ODBC Data Driver installed. The file location is:

```
SYS:PUBLIC\SQLC\samples\windows\access\demo.mdb
```

Once the file is copied, the following steps should be performed:

1. On the database server (third tier), create an ODBC Data Source Name "demo" that uses the Microsoft Access ODBC driver and the database file demo.mdb.
2. Run the Data Source Administrator and create a Data Source named "demo" on the network server (second tier) that points to the ODBC Access data source "demo" on the database server (third tier). See the *Administration Guide* for details about this step.

Warning: If the Access database is on a NetWare mounted drive, then the NetWare login and the Windows NT login must be the same username.

The Access database is now available for use with the sample programs.

2.4.3 Using jdbcping to Verify a Connection

The jdbcping program can be executed in an MS–DOS command window or at a NetWare console. A Java runtime environment must be installed, and the files jdbcping.class and sqlcjdbc.jar must be in the CLASSPATH as discussed above for Windows or NetWare.

The program jdbcping is used as follows:

```
Usage:   java jdbcping <url> <username> <password>
Example: java jdbcping jdbc:sqli://@server1:1958 -
                                        /demo myuser mypass
```

If the JDBC connection is established, this message is displayed:

```
Successful Connection
```

An alternate form of usage is to include the username and password in the URL:

```
Usage:   java jdbcping <url> "" ""
Example: java jdbcping jdbc:sqli://myuser:mypass@server1:1958 -
                                        /demo "" ""
```

2.4.4 Using jdbcjoin to Retrieve Data

After a connection has been verified, the sample data may be retrieved using the jdbcjoin program, which also must be in the CLASSPATH for Windows or NetWare.

The jdbcjoin program is used as follows:

```
Usage:   java jdbcjoin <url> <username> <password>
Example: java jdbcjoin jdbc:sqli://@server1:1958 -
                                        /demo myuser mypass
```

An alternate form of usage is to include the username and password in the URL:

```
Usage:   java jdbcjoin <url> "" ""
Example: java jdbcjoin jdbc:sqli://myuser:mypass@server1:1958 -
                                        /demo "" ""
```

If the data is successfully retrieved, the following text is displayed:

```
select
  dept.deptno, dept.dname, emp.empno, emp.ename, emp.ssno, emp.jobcls
from
  dept, emp
where
  dept.deptno = emp.deptno and dept.deptno < 3

Dept.Deptno  Dept.Dname  Emp.Empno  Emp.Ename      Emp.SSNO    Emp.JobCls
-------------------------------------------------------------------------
     1       MARKETING    20265     BASINGER_R     394-90-3583     2
     1       MARKETING    20337     GILBREATH_R    553-40-5418     5
     1       MARKETING    20365     MILLER_R       248-95-1471     6
     1       MARKETING    32024     GOVE_T         742-35-6859     4
     1       MARKETING    33297     ELLISON_W      252-60-1283     2
     1       MARKETING    33733     BANFIELD_S     688-21-4136     4
     1       MARKETING    33771     NASER_K        848-17-0910     8
     1       MARKETING    34410     MCCORRY_K      015-84-9007     7
     1       MARKETING    35844     MCCARTNEY_H    211-02-7599     7
     1       MARKETING    35909     LACY_C         494-12-0675     7
     1       MARKETING    38342     KINGSLEY_S     107-42-3232     5
     1       MARKETING    39748     BARNETT_N      179-22-5607     4
     1       MARKETING    63297     ELLISON_V      152-60-1283     1
     1       MARKETING    63647     HAWTHORNE_A    735-64-9259     3
     1       MARKETING    63910     BRANNEN_P      355-08-2564     3
```

```
    1       MARKETING    64689     FULFORD_W      546-61-8895     5
    1       MARKETING    67528     BLOCK_F        084-40-0022     3
    1       MARKETING    68355     SWIFT_E        358-51-5799     9
    2       TRAINING     20384     KNAPP_L        381-26-6148     3
    2       TRAINING     31028     ROCHE_D        671-74-2192     9
    2       TRAINING     32482     NICHOLL_F      129-79-0929     8
    2       TRAINING     34948     ELLINGTON_A    686-40-9928     4
    2       TRAINING     35038     ROMANN_H       079-54-8297     9
    2       TRAINING     39208     ELDER_N        460-95-5451     2
    2       TRAINING     64948     ELLINGTON_Z    586-40-9928     3
    2       TRAINING     68551     HOLMES_P       664-23-5820     3
    2       TRAINING     69208     ELDER_M        360-95-5451     1
-------------------------------------------------------------------------
```

3

JDBC Implementation

3.1 Introduction

The JDBC API specifies a level of functionality that must be provided by a JDBC driver in order to connect to JDBC applications. This level applies to a set of classes and methods and a set of datatypes.

There is a set of JDBC driver and connection methods that allow a JDBC application to *discover* the functionality available from the JDBC driver. The JDBC application then knows the limits of the driver and does not try to exceed the limits. The results of these methods for the SQL *C*–JDBC are documented in Appendix A. A complete discussion of the driver and connection methods can be found by browsing the documentation at `http://java.sun.com/products/jdbc/index.html`.

3.2 SQL Grammar

SQL *Connector* supports most all of the elements in the minimum SQL grammar, and some elements of the core and extended SQL grammar. For a complete discussion of SQL grammar elements, see Appendix C in the *Microsoft ODBC 2.0 Programmer's Reference and SDK Guide.*

Core elements not supported are Data Definition Language (DDL) statements (`CREATE` and `DROP TABLE`, `CREATE` and `DROP INDEX`). SQL *Connector* uses database table and column information that is imported from existing physical databases. Tables and indexes are first created in the physical database using the appropriate database tools, then imported into SQL *Connector*.

The SQL Grammar elements are categorized as follows:

1. Minimum SQL grammar elements:
   - Data Manipulation Language (DML): simple `SELECT`, `INSERT`, `UPDATE` and `DELETE`
   - Expressions: simple, such as `A > B + C`
   - Data type: `CHAR`
2. Core SQL grammar elements:
   - Minimum SQL grammar and data types
   - DML: full `SELECT`
   - Expressions: subquery and aggregates such as `SUM` and `MIN`
   - Data types: `DECIMAL`, `SMALLINT`, `FLOAT`, `REAL`, `DOUBLE`
3. Extended SQL grammar
   - Minimum and Core SQL grammar and data types
   - Expressions: scalar functions such as `SUBSTRING` and date, time, and timestamp literals
   - Data types: `DATE`, `TIME`, `TIMESTAMP`

3.3 SQL Datatypes

SQL *C*–JDBC supports the following SQL data types:

SQL *Connector* Datatype	Java Type	Maximum Column Size	Nullable	Case Sensitive	Searchable
char	String	32000	Y	Y	Y
varchar	String	32000	Y	Y	Y
decimal	java.math.BigDecimal	19	Y	N	Y
tinyint	byte	3	Y	N	Y
smallint	short	5	Y	N	Y
integer	int	10	Y	N	Y
bigint	long	19	Y	N	Y
real	float	7	Y	N	Y
double	double	16	Y	N	Y
date	date	~	Y	N	Y
timestamp	timestamp	~	Y	N	Y

A

Driver and Connection Attributes

The following table lists the information type and values that are returned by Driver and Connection MetaData functions. A complete discussion of the driver and connection methods can be found by browsing the documentation at http://java.sun.com/products/jdbc/index.html.

Information Type	Information Value
Driver Information	
acceptsURL	jdbc:sqli
getClass.getName	sqli.sql.sqliDriver
getMajorVersion	1
getMinor Version	0
jdbcCompliant	false
Driver Property[0].name	service
Driver Property[0].description	SQL Connector Server program
Driver Property[0].value	vtx16
Driver Property[0].required	false
Driver Property[0].choices	null
Driver Property[1].name	fetch_size
Driver Property[1].description	Fetch buffer size
Driver Property[1].value	8192
Driver Property[1].required:	false
Driver Property[1].choices	null
Connection Information	
getAutoCommit	true
isReadOnly	false
getTransactionIsolation (default)	TRANSACTION_REPEATABLE_READ
MetaData Information	
allProceduresAreCallable	false
allTablesAreSelectable	true
dataDefinitionCausesTransactionCommit	false
doesMaxRowSizeIncludeBlobs	true
getCatalogSeparator	null
getCatalogTerm	null
getDatabaseProductName	SQL Connector
getDatabaseProductVersion	3.1
getDefaultTransactionIsolation	4
getDriverMajorVersion	2
getDriverMinorVersion	3
getDriverName	sqli.sql.sqliDriver
getDriverVersion	2.3

Information Type	**Information Value**
getExtraNameCharacters	null
getIdentifierQuoteString	"
getMaxBinaryLiteralLength	0
getMaxCatalogNameLength	0
getMaxCharLiteralLength	32767
getMaxColumnNameLength	31
getMaxColumnsInGroupBy	8
getMaxColumnsInIndex	8
getMaxColumnsInOrderBy	8
getMaxColumnsInSelect	1023
getMaxColumnsInTable	1023
getMaxConnections	0
getMaxCursorNameLength	0
getMaxIndexLength	120
getMaxProcedureNameLength	0
getMaxRowSize	8192
getMaxSchemaNameLength	31
getMaxStatementLength	32767
getMaxStatements	128
getMaxTableNameLength	31
getMaxTablesInSelect	30
getMaxUserNameLength	0
getNumericFunctions	ABS,INT2,INT4,FLOAT4,FLOAT8,MOD,ACOS, ASIN,ATAN,COS,COT,DEGREES,EXP,LOG, LOG10,MOD,PI,POWER,RADIANS,RAND, ROUND,SIGN,SIN,SQRT,TAN,TRUNCATE
getProcedureTerm	null
getSQLKeywords	<see Appendix B>
getSchemaTerm	schema
getSearchStringEscape	\
getStringFunctions	CONCAT, STRING, SUBSTRING, TO_CHAR, ASCII, CHAR, INSERT, LCASE, LEFT, LENGTH,LOCATE, LTRIM, REPEAT, REPLACE, RIGHT, RTRIM, SPACE, UCASE
getSystemFunctions	NVL, ISNULL, IFNULL, DATABASE, USER
getTimeDateFunctions	TO_DATE, CURDATE, CURTIME, DAYNAME, DAYOFMONTH, DAYOFWEEK, DAYOFYEAR, HOUR, MINUTE, MONTH, MONTHNAME, NOW, QUARTER, SECOND, TIMESTAMPADD, TIMESTAMPDIFF, WEEK, YEAR
getUserName	<username>
isCatalogAtStart	true
isReadOnly	false
nullPlusNonNullIsNull	true
nullsAreSortedAtEnd	false
nullsAreSortedAtStart	false

Information Type	Information Value
nullsAreSortedHigh	true
nullsAreSortedLow	false
storesLowerCaseIdentifiers	true
storesLowerCaseQuotedIdentifiers	true
storesMixedCaseIdentifiers	false
storesMixedCaseQuotedIdentifiers	false
storesUpperCaseIdentifiers	false
storesUpperCaseQuotedIdentifiers	false
supportsANSI92EntryLevelSQL	true
supportsANSI92FullSQL	true
supportsANSI92IntermediateSQL	true
supportsAlterTableWithAddColumn	false
supportsAlterTableWithDropColumn	false
supportsCatalogsInDataManipulation	false
supportsCatalogsInIndexDefinitions	false
supportsCatalogsInPrivilegeDefinitions	false
supportsCatalogsInProcedureCalls	false
supportsCatalogsInTableDefinitions	false
supportsColumnAliasing	false
supportsConvert	false
supportsCoreSQLGrammar	true
supportsCorrelatedSubqueries	true
supportsDataDefinitionAnd DataManipulationTransactions	true
supportsDataManipulationTransactionsOnly	false
supportsDifferentTableCorrelationNames	false
supportsExpressionsInOrderBy	true
supportsExtendedSQLGrammar	true
supportsFullOuterJoins	true
supportsGroupBy	true
supportsGroupByBeyondSelect	false
supportsGroupByUnrelated	true
supportsIntegrityEnhancementFacility	false
supportsLikeEscapeClause	true
supportsLimitedOuterJoins	true
supportsMinimumSQLGrammar	true
supportsMixedCaseIdentifiers	true
supportsMixedCaseQuotedIdentifiers	false
supportsMultipleResultSets	false
supportsMultipleTransactions	true
supportsNonNullableColumns	true
supportsOpenCursorsAcrossCommit	false
supportsOpenCursorsAcrossRollback	false
supportsOpenStatementsAcrossCommit	true
supportsOpenStatementsAcrossRollback	true

Information Type	**Information Value**
supportsOrderByUnrelated	true
supportsOuterJoins	true
supportsPositionedDelete	false
supportsPositionedUpdate	false
supportsSchemasInDataManipulation	true
supportsSchemasInIndexDefinitions	true
supportsSchemasInPrivilegeDefinitions	false
supportsSchemasInProcedureCalls	false
supportsSchemasInTableDefinitions	true
supportsSelectForUpdate	true
supportsStoredProcedures	false
supportsSubqueriesInComparisons	true
supportsSubqueriesInExists	true
supportsSubqueriesInIns	true
supportsSubqueriesInQuantifieds	false
supportsTableCorrelationNames	true
supportsTransactionIsolationLevel TRANSACTION_NONE	false
supportsTransactionIsolationLevel TRANSACTION_READ_UNCOMMITTED	false
supportsTransactionIsolationLevel TRANSACTION_READ_COMMITTED	false
supportsTransactionIsolationLevel TRANSACTION_REPEATABLE_READ	true
supportsTransactionIsolationLevel TRANSACTION_SERIALIZABLE	false
supportsTransactions	true
supportsUnion	false
supportsUnionAll	false
usesLocalFilePerTable	false
usesLocalFiles	false

B

SQL Keywords

The following keywords are reserved for use by SQL *Connector*. If there is existing SQL metatdata (tables and columns) that already use these keywords, then references to the metatdata must be surrounded by double quotes. If new metadata (tables and columns) is created, then the metadata should not use words in this list.

abort	att_name	audit	average	bcd	bcdfixed
bcdflt	binary	bottom	breakable	call	carrycolumns
center	checkpoint	columnno	committed	concat	copyin
copyout	cumulative	currentdate	currentmoment	currenttime	database
databasename	datatype	days	daytime	dba	detail
detailno	editstring	elseif	enddo	endif	endpage
endsection	endstore	endstream	errorlimit	exclusive	exit
fetchnumber	fieldcomment	fieldinfo	fieldlabel	filestatus	fixed
float4	float8	footsize	format	forms	function
getdate	gettime	giving	groupcount	groupno	guide
head	header	headformat	heading	highlighting	host
hostfield	hostparameter	hours	ifnull	index	initpage
initsection	inout	inquire	int1	int2	int4
int8	isnull	killdbin	leave	leftmargin	lineno
lowercase	maximum	message	midnight	minimum	minlocks
minutes	months	need	none	nvl	oddpage
outerjoin	overwriting	page	pagebottom	pagelength	pageno
pagetop	paging	panelwidth	parameter comment	presorted	print
prompt	prompting	protected	reconfigure	recordno	recordtype
recover	rel_id	rel_name	repeatable	report	reportbottom
reporttop	reserving	return	returns	rightmargin	savepoint
seconds	sectionno	segmentsize	separation	serializable	shared
show	skip	skipping	space	spacing	spreading
start	startpage	startsection	status	stddev	store
stream	sync	synonym	to_char	to_date	to_number
today	top	total	truncate	unbreakable	uncommitted
unsafe	unset	uppercase	userid	variance	vaxday
vaxhour	vaxmidnight	vaxminute	vaxsecond	vaxtime	verify
week	weeks	while	width	years	

C

JDBC Errors

When an error occurs, the SQL *C*–JDBC driver returns an sqliException and an error message. The SQL *C*–JDBC driver gets this information from errors that are detected by the driver itself and from errors that are returned by SQL *C*–DRB.

For errors that occur within the Data Source or from data sources connected to the Data Source, the SQL *C*–JDBC driver returns the message identification and the message text. For a list of native errors, see the table "sqlmsg" in the server system Data Source named msgdb.dic. This table has four columns:

```
msgnum      integer not null
facility    char (8) not null
msgid       char (26) not null
msgtxt      char (160) not null
```

The message identification and message text are the third and fourth columns.

Error Message Syntax

Error messages have the following format:

```
sqli.sql.sqlidbException: error_message
```

The error message returned by SQL *C*–DRB is the concatenation of the message identification and the message text. There may be more than one error line returned.

Example Error Messages

If the JDBC data source is not found:

```
sqli.sql.sqliDriver$sqliInvalidPortException: Invalid port in URL: null
```

If the Data Source is not found:

```
sqli.sql.sqlidbException: %RQP-E-DICPARSE, Can't Parse Dictionary
                                File Name '../examples/xxx'
```

If a table in the Data Source is not found:

```
sqli.sql.sqlidbException: %RQP-E-TABUND, Table empx Undefined in
                                Dictionary File ../examples/demo
```

D

JDBCPING Source Code

```
//  JDBC Ping Test

import java.util.*;
import java.sql.*;

//--------------------------------------------------------------------------
public class jdbcping
{
  public static void main (String args[]) throws Exception // Throwable
  {
  // url = arg[0], uid = arg[1], pwd = arg[2]
  if (args.length < 3) {
    System.out.println ("Usage: jdbcping url uid pwd");
    }
  else {
    String url = args[0];
    String uid = args[1];
    String pwd = args[2];
    try {
  Driver d = (Driver)Class.forName("sqli.sql.sqliDriver").newInstance();
      try {
        System.out.println ("\nConnecting to JDBC Data Source");
        System.out.println ("  url = " + url);
        System.out.println ("  uid = " + uid);
        System.out.println ("  pwd = " + pwd);
        Connection c = DriverManager.getConnection (url, uid, pwd);
        System.out.println ("\nSuccessful Connection\n");
        }
      catch (Exception e) {
        System.out.println (e);
        }
      }
    catch (Exception e) {
      System.out.println (e);
      }
    }
  }
}
```

E

JDBCJOIN Source Code

```
//  JDBC Join Test

import java.util.*;
import java.sql.*;

//---------------------------------------------------------------------

public class jdbcjoin
{
  public static void main (String args[]) throws Exception // Throwable
  {
  Driver     d = null;
  Connection c = null;
  Statement  s = null;
  ResultSet  r = null;
  boolean    more;

  // url = arg[0], uid = arg[1], pwd = arg[2]
  if (args.length < 3) {
    System.out.println ("Usage: jdbcjoin url user pwd");
    System.exit (0);
    }
  String url = args[0];
  String uid = args[1];
  String pwd = args[2];
  try {
    d = (Driver)Class.forName("sqli.sql.sqliDriver").newInstance();
    try {
      Connection c = DriverManager.getConnection (url, uid, pwd);
    }
    catch (Exception e) {
      System.out.println (e);
      System.exit (0);
      }
    }
  catch (Exception e) {
    System.out.println (e);
    System.exit (0);
    }

  System.out.println ("\nselect");
  System.out.println (" dept.deptno, dept.dname, emp.empno, emp.ename,
                        emp.ssno, emp.jobcls");
  System.out.println ("from");
  System.out.println (" dept, emp");
  System.out.println ("where");
  System.out.println (" dept.deptno = emp.deptno and dept.deptno < 3\n");
```

```
  System.out.println ("Dept.Deptno  Dept.Dname Emp.Empno  Emp.Ename " +
                      "Emp.SSNO    Emp.JobCls");
  System.out.println ("-----------------------------------------------");

  s = c.createStatement ();
  r = s.executeQuery ("select d.deptno, d.dname, e.empno, e.ename, " +
                  "e.ssno, e.jobcls from dept d, emp e " +
                  "where dept.deptno = emp.deptno and dept.deptno < 3");
  more = r.next ();
  while (more) {
    StringBuffer buf = new StringBuffer();
    buf.append ("     ");
    buf.append (getData (r, 1,  8));
    buf.append (getData (r, 2, 13));
    buf.append (getData (r, 3, 10));
    buf.append (getData (r, 4, 15));
    buf.append (getData (r, 5, 16));
    buf.append (getData (r, 6,  7));
    System.out.println (buf);
    more = r.next ();
    }  // end fetch
  System.out.println ("-----------------------------------------------");

  r.close ();
  s.close ();
  c.close ();
  } // end main

//---------------------------------------------------------------------
public static StringBuffer getData (ResultSet rs, int colNo, int outLen)
                                     throws Exception
  {
  StringBuffer b = new StringBuffer ();
  b.append (rs.getString (colNo));
  int ind = b.length();
  if (b.length() == 0) {
    b.append("~");
    ind = 1;
    }
  else {
    for (int i=ind; i<outLen; i++)
      b.append(" ");
    }
  return b;
  }

} // end class jdbcjoin
```

F

Programming Notes

F.1 SQL Data Types

F.1.1 SQL Double Datatype Restriction

The maximum value of an SQL Double datatype is `1.0e+126`. The SQL-92 documented maximum value is `1.797693134862316e+308`.