Progress DataDirect

Progress DataDirect for JDBC for Informix

User's Guide

Release 6.0.0

Copyright

Table of Contents

1

Welcome to the Progress DataDirect for JDBC for Informix Driver

The Progress® DataDirect® for JDBC™ for Informix™ driver (Informix driver) supports the JDBC API for SQL read-write access to Informix Dynamic Server.

The documentation for the driver also includes the *Progress DataDirect for JDBC Drivers Reference*. The reference provides general reference information for all DataDirect drivers for JDBC, including content on troubleshooting, supported SQL escapes, and DataDirect tools.

For the complete documentation set, visit the Progress DataDirect Connectors Documentation Hub: https://docs.progress.com/category/datadirect-ibm-informix.

For details, see the following topics:

- Additional information
- Contacting Technical Support

What's new in this release?

Support and certification

Visit the following web pages for the latest support and certification information.

- Release Notes: https://www.progress.com/datadirect-connectors/whats-new#jdbc
- DataDirect Product Compatibility Guide: https://docs.progress.com/bundle/datadirect-product-compatibility/resource/datadirect-product-compatibility.pdf

Changes Since 6.0.0 GA

- **Enhancements**
  - The driver has been enhanced to comply with FIPS standards for data encryption. As part of this enhancement, the driver was tested with FIPS 140-3 enabled using a Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance. See FIPS (Federal Information Processing Standard) on page 46 for details.
- **Changed Behavior**
  - The connection property `SpyAttributes` has been updated to exclude the attribute `load=classname`, which was previously used to load the driver specified by the given class name. See SpyAttributes on page 88 for details.

Changes for 6.0.0 GA

- **Driver Enhancements**
  - The driver has been enhanced to include timestamp in the Spy and JDBC packet logs by default. If required, you can disable the timestamp logging by specifying the following at connection: For Spy logs, set `spyAttributes=(log=(file)Spy.log;timestamp=no)` and for JDBC packet logs, set `ddtdbg.ProtocolTraceShowTime=false`.
  - Interactive SQL for JDBC (JDBCISQL) is now installed with the product. JDBCISQL is a command-line interface that supports connecting your driver to a data source, executing SQL statements and retrieving results in a terminal. This tool provides a method to quickly test your drivers in an environment that does not support GUIs. See Interactive SQL for JDBC (JDBCISQL) on page 29 for details.
  - The RegisterStatementPoolMonitorMBean connection property has been added. Note that the driver no longer registers the Statement Pool Monitor as a JMX MBean by default. You must set RegisterStatementPoolMonitorMBean to `true` to register the Statement Pool Monitor and manage statement pooling with standard JMX API calls. See RegisterStatementPoolMonitorMBean on page 85 for details.
- **Changed Behavior**
  - If you attempt to execute a DatabaseMetadata API with catalogName filter set as `''` (Empty String) against Informix 14.1, the API will return the full list of objects instead of an empty result. All DatabaseMetadata APIs, such as getTables() and getColumns(), which support catalogName as filter are affected by this behavior.

Requirements

The driver is compatible with JDBC 2.0, 3.0, and 4.0.

The driver requires a Java Virtual Machine (JVM) that is Java SE 8 or higher, including Oracle JDK, OpenJDK, and IBM SDK (Java) distributions.

Installing and setting up the driver

This section provides you with an overview of the steps required to install and set-up the driver. After completing this procedure, you will be able to begin accessing data with your application.

To begin accessing data with the driver:

1. Install the driver:
   a) After downloading the product, unzip the installer files to a temporary directory.
   b) From the installer directory, run the appropriate installer file to start the installer.
      - **Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.exe`
      - **Non-Windows**: `PROGRESS_DATADIRECT_JDBC_INSTALL.jar`
   c) Follow the prompts to complete installation.

   The installer program supports multiple installation methods, including command-line and silent installations. For detailed instructions, refer to the *Progress DataDirect for JDBC Drivers Installation Guide*.
2. Set your system CLASSPATH to include the driver `.jar` file. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. The following examples demonstrate setting the CLASSPATH from a command line using the default installation directory.
   - **Windows Example**

     ```
     CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\informix.jar
     ```
   - **UNIX/LINUX Example**

     ```
     CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/informix.jar
     ```
3. Configure your driver using one of the following methods:
   - **Connection URL**: You can begin using the driver immediately by passing a connection URL with your application or tool. The following example shows how to connect using user ID/password authentication.

     ```
     jdbc:datadirect:informix://myserver:1526;InformixServer=myinformixserver;
     DatabaseName=account;User=jsmith;Password=secret;
     ```

     Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

     Note: See Authentication on page 45 for details.

- **Data sources**: The driver also supports connecting using JDBC data sources. A JDBC data source is a Java object, specifically a DataSource object, that defines connection information required for a JDBC driver to connect to the database. See Connecting using data sources for more information.

Note: For most connections, specifying the minimum required connection properties is sufficient to begin accessing data; however, you can provide values for optional properties to use additional supported features and improve performance.

4. Set the values for any optional properties that you want to configure. For additional information on optional features and functionality, see the following resources:
   - Connection URL examples provides connection string examples that can be used to configure common functionality and features. You can modify and combine these examples to create a string that best suites your environment.
   - Connection property descriptions provides a complete list of supported properties by functionality.
   - Performance considerations describes connection properties that affect performance, along with recommended settings.
5. Connect to your service and begin accessing data with your applications, BI tools, database tools, and more. To help you get started, the following resources guide you through accessing data with some common tools:
   - Interactive SQL for JDBC (JDBCISQL): JDBCISQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal. This tool provides a method of quickly testing your driver in an environment that does not support GUIs.
   - Tableau: Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data.
   - DbVisualizer: DB Visualizer is a database tool that allows you to connect and execute SQL statements against your data.
   - DataDirect Test: DataDirect Test allows you to test connect, execute SQL statements, and practice using the JDBC API right out of the box.

This completes the deployment of the driver.

Driver and DataSource classes

The following are the `Driver` and `DataSource` classes used by the driver:

Driver class:

com.ddtek.jdbc.informix.InformixDriver

DataSource class:

com.ddtek.jdbcx.informix.InformixDataSource

Connection URL examples

After setting the CLASSPATH, the connection information needs to be passed in the form of a connection URL. This section provides examples of connection strings configured to use common features and functionality. You can modify and/or combine these examples to create a connection string for your environment.

Note:

- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: {*value*}. For example: `User={hello }` or `Password={{hello}}`.

User ID/password authentication

This example includes the properties used to connect with the user ID/password authentication.

```
jdbc:datadirect:informix://servername:port;
InformixServer=informixserver;DatabaseName=databasename;
User=userID;Password=password;[property=value[;...]];
```

where:

servername

is the IP address or name of the computer that is running the database server to which you want to connect.

port

is the number of the TCP/IP port.

informixserver

is the name of the database server to which you want to connect.

databasename

is the name of the database to which you want to connect.

userID

specifies the user ID that is used to connect to the Informix database.

password

specifies a password that is used to connect to your Informix database.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

The following example connection string includes the properties required for connecting with the user ID/password authentication.

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:informix://myserver:1526;
  InformixServer=myinformixserver;DatabaseName=account;
  User=jsmith;Password=secret;");
```

See also

Connection property descriptions on page 57
Authentication on page 45

Failover

This string configures the driver to use connection failover in conjunction with connection retry.

```
jdbc:datadirect:informix://servername:port;InformixServer=informixserver;
DatabaseName=databasename;User=userID;Password=password;
AlternateServers=(alternateserver);
ConnectionRetryCount=connectionretrycount;ConnectionRetryDelay=connectionretrydelay;
[property=value[;...]];
```

where:

servername

is the IP address or name of the primary server.

port

is the number of the TCP/IP port of the primary server.

informixserver

is the name of the Informix database server to which you want to connect.

databasename

is the name of the database in the primary server to which you want to connect.

userID

specifies the user ID that is used to connect to the Informix database.

password

specifies a password that is used to connect to your Informix database.

alternateserver

specifies a list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. This example uses the default failover method, connection failover.

connectionretrycount

specifies the number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.

connectionretrydelay

specifies the number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

The following example configures the driver to use connection failover in conjunction with connection retry.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:informix://myserver1:2003;InformixServer=myinformixserver1;
    DatabaseName=payroll;User=jsmith;Password=secret;
    AlternateServers=(myserver2:2003;InformixServer=myinformixserver2,myserver3:2003);
    ConnectionRetryCount=2;ConnectionRetryDelay=5;");
```

See also

Connection property descriptions on page 57
Failover on page 46

Data Types

The following table lists the data types supported by the Informix driver and how they are mapped to the JDBC data types.

Table 1: Informix Data Types

Informix Data Type	JDBC Data Type
BLOB	BLOB
BOOLEAN	BOOLEAN
BYTE	LONGVARBINARY
CHAR	CHAR
CLOB	CLOB
DATE	DATE
DATETIME HOUR TO SECOND	TIME

Informix Data Type	JDBC Data Type
DATETIME YEAR TO DAY	DATE
DATETIME YEAR TO FRACTION(5)	TIMESTAMP
DATETIME YEAR TO SECOND	TIMESTAMP
DECIMAL	DECIMAL
FLOAT	FLOAT
INT8	BIGINT
INTEGER	INTEGER
LVARCHAR	VARCHAR
MONEY	DECIMAL
NCHAR	CHAR or NCHAR[1]
NVARCHAR	VARCHAR or NVARCHAR[1]
SERIAL	INTEGER
SERIAL8	BIGINT
SMALLFLOAT	REAL
SMALLINT	SMALLINT
TEXT	LONGVARCHAR
VARCHAR	VARCHAR

getTypeInfo()

The following table provides getTypeInfo() results for all Informix databases supported by the Informix driver.

[1] When JDBCBehavior=1, the first value applies and when JDBCBehavior=0, the second value applies.

Table 2: getTypeInfo() for Informix

TYPE_NAME = blob AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 2004 (BLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = blob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = boolean AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 16 (BOOLEAN) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = boolean MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 1 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = byte AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -4 (LONGVARBINARY) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = byte MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = char AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = 1 (CHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = char MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32766 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = clob AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = 2005 (CLOB) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = clob MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = date AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = false LITERAL_PREFIX = {d ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = date MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = datetime hour to second AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 92 (TIME) FIXED_PREC_SCALE = false LITERAL_PREFIX = {t ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = datetime hour to second MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 8 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime year to day AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 91 (DATE) FIXED_PREC_SCALE = false LITERAL_PREFIX = {d ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = datetime year to day MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = datetime year to fraction(5) AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = datetime year to fraction(5) MAXIMUM_SCALE = 5	MINIMUM_SCALE = 5 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 25 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = datetime year to second AUTO_INCREMENT = NULL CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 93 (TIMESTAMP) FIXED_PREC_SCALE = false LITERAL_PREFIX = {ts ' LITERAL_SUFFIX = '} LOCAL_TYPE_NAME = datetime year to second MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = decimal AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *precision*,*scale* DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = decimal MAXIMUM_SCALE = 32	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 32 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = float AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 6 (FLOAT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = float MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 15 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

TYPE_NAME = int8 AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = int8 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = integer AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = integer MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = lvarchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL (Informix 9.2, 9.3), *max length* (Informix 9.4, 10) DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = lvarchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2048 (Informix 9.2, 9.3), 32739 (Informix 9.4, 10) SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

TYPE_NAME = money AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = *precision*,*scale* DATA_TYPE = 3 (DECIMAL) FIXED_PREC_SCALE = true LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = money MAXIMUM_SCALE = 32	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 32 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = nchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *length* DATA_TYPE = 1 (CHAR) [2] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = nchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 32766 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = nvarchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *max length* DATA_TYPE = 12 (VARCHAR) [3] FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = nvarchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 254 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

[2] If JDBCBehavior=0, the value returned for DATA_TYPE is -15 (NCHAR). If JDBCBehavior=1, the value returned for DATA_TYPE is 1 (CHAR).

[3] If JDBCBehavior=0, the value returned for DATA_TYPE is -9 (NVARCHAR). If JDBCBehavior=1, the value returned for DATA_TYPE is 12 (VARCHAR).

TYPE_NAME = serial AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = start DATA_TYPE = 4 (INTEGER) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = serial MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 10 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = serial8 AUTO_INCREMENT = true CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = -5 (BIGINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = serial8 MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 19 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = smallfloat AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 7 (REAL) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallfloat MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 7 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false

TYPE_NAME = smallint AUTO_INCREMENT = false CASE_SENSITIVE = false CREATE_PARAMS = NULL DATA_TYPE = 5 (SMALLINT) FIXED_PREC_SCALE = false LITERAL_PREFIX = NULL LITERAL_SUFFIX = NULL LOCAL_TYPE_NAME = smallint MAXIMUM_SCALE = 0	MINIMUM_SCALE = 0 NULLABLE = 1 NUM_PREC_RADIX = 10 PRECISION = 5 SEARCHABLE = 2 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = false
TYPE_NAME = text AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = NULL DATA_TYPE = -1 (LONGVARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = text MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 2147483647 SEARCHABLE = 0 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL
TYPE_NAME = varchar AUTO_INCREMENT = NULL CASE_SENSITIVE = true CREATE_PARAMS = *max length* DATA_TYPE = 12 (VARCHAR) FIXED_PREC_SCALE = false LITERAL_PREFIX = ' LITERAL_SUFFIX = ' LOCAL_TYPE_NAME = varchar MAXIMUM_SCALE = NULL	MINIMUM_SCALE = NULL NULLABLE = 1 NUM_PREC_RADIX = NULL PRECISION = 254 SEARCHABLE = 3 SQL_DATA_TYPE = NULL SQL_DATETIME_SUB = NULL UNSIGNED_ATTRIBUTE = NULL

SQL escape sequences

The driver supports the following SQL escape sequences.

- Date, Time, and Timestamp Escape Sequences
- Scalar Functions
- Outer Join Escape Sequences
- LIKE Escape Character Sequence for Wildcards

Refer to "SQL escape sequences" in the *Progress DataDirect for JDBC Drivers Reference* for information about SQL escape sequences.

Supported scalar functions

You can use scalar functions in SQL statements with the following syntax:

`{fn scalar-function}`

where:

scalar-function

is a scalar function supported by the driver, as listed in the following table.

Example:

```
SELECT id, name FROM emp WHERE name LIKE {fn UCASE('Smith')}
```

Table 3: Supported Scalar Functions

String Functions	Numeric Functions	Timedate Functions	System Functions
CONCAT	ABS	CURDATE	USERNAME
LEFT	ACOS	CURTIME	DBNAME
LENGTH	ASIN	DAYOFMONTH	IFNULL
LTRIM	ATAN	DAYOFWEEK	
REPLACE	ATAN2	MONTH	
RTRIM	COS	NOW	
SUBSTRING	COT	TIMESTAMPADD	
	EXP	TIMESTAMPDIFF	
	FLOOR	YEAR	
	LOG		
	LOG10		
	MOD		

String Functions	Numeric Functions	Timedate Functions	System Functions
	PI POWER ROUND SIN SQRT TAN TRUNCATE		

DataDirect tools

Progress DataDirect for JDBC drivers install the set of tools described in this section. For detailed instructions on using these tools, refer to the corresponding topics in the *Progress DataDirect for JDBC Drivers Reference*.

- DataDirect Test allows you to test your JDBC driver and learn the JDBC API.
- DataDirect Connection Pool Manager allows you to pool connections when accessing databases. When your applications use connection pooling, connections are reused rather than created each time a connection is requested. Because establishing a connection is among the most costly operations an application may perform, using Connection Pool Manager to implement connection pooling can significantly improve performance.
- Statement Pool Monitor loads statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance.
- DataDirect Spy logs detailed information about calls your driver makes that can be used for troubleshooting.

Troubleshooting

The *Progress DataDirect for JDBC Drivers Reference* provides information on troubleshooting problems should they occur. Refer to the "Troubleshooting" section in the *Reference* for details.

Additional information

In addition to the content provided in this guide, the documentation set also contains detailed conceptual and reference information that applies to all the drivers. For more information in these topics, refer the *Progress DataDirect for JDBC Drivers Reference* or use the links below to view some common topics:

- "JDBC support" describes support for JDBC interfaces and methods for the Progress DataDirect for JDBC drivers.
- "JDBC extensions" describes the JDBC extensions provided by the com.ddtek.jdbc.extensions package.
- "SQL escape sequences for JDBC" provides an overview of SQL escape sequences for JDBC. In addition, it documents the scalar functions that you use in SQL statements.
- "Security best practices for JDBC applications" describes the security best practices you should employ when developing and deploying your application with the driver.

Contacting Technical Support

Progress DataDirect offers a variety of options to meet your support needs. Please visit our Web site for more details and for contact information:

https://www.progress.com/support

The Progress DataDirect Web site provides the latest support information through our global service network. The SupportLink program provides access to support contact details, tools, patches, and valuable information, including a list of FAQs for each product. In addition, you can search our Knowledgebase for technical bulletins and other information.

When you contact us for assistance, please provide the following information:

- Your number or the serial number that corresponds to the product for which you are seeking support, or a case number if you have been provided one for your issue. If you do not have a SupportLink contract, the SupportLink representative assisting you will connect you with our Sales team.
- Your name, phone number, email address, and organization. For a first-time call, you may be asked for full information, including location.
- The Progress DataDirect product and the version that you are using.
- The type and version of the operating system where you have installed your product.
- Any database, database version, third-party software, or other environment information required to understand the problem.
- A brief description of the problem, including, but not limited to, any error messages you have received, what steps you followed prior to the initial occurrence of the problem, any trace logs capturing the issue, and so on. Depending on the complexity of the problem, you may be asked to submit an example or reproducible application so that the issue can be re-created.
- A description of what you have attempted to resolve the issue. If you have researched your issue on Web search engines, our Knowledgebase, or have tested additional configurations, applications, or other vendor products, you will want to carefully note everything you have already attempted.
- A simple assessment of how the severity of the issue is impacting your organization.

2

Tutorials

The following sections guide you through using the driver to access your data with some common third-party applications. For information on installing your driver and setting the CLASSPATH, see "Installing and setting-up the driver."

For details, see the following topics:

- Interactive SQL for JDBC (JDBCISQL)
- Tableau
- DbVisualizer

Interactive SQL for JDBC (JDBCISQL)

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Interactive SQL for JDBC (JDBCISQL). JDBCISQL supports a command line interface that allows you to connect to a data source, execute SQL statements and retrieve results for display on a terminal.

To execute commands with JDBCISQL:

1. Start the ISQL tool. Do one of the following:
   - On Windows, double-click the`jdbcisql.bat` file in the *`install_dir`*`\jdbcisql` folder. Or, from a command prompt, navigate to the *`install_dir`*`\jdbcisql` directory and run the `jdbcisql.bat` file.

- On Linux and UNIX, change to the *install_dir*\isql directory and run jdbcisql.sh.

The Interactive SQL prompt appears.

2. Type the driver name class; then, press **Enter**:

```
com.ddtek.jdbc.informix.InformixDriver
```

3. Type connect followed by the connection URL for the driver; then, press **Enter**. For example:

```
connect jdbc:datadirect:informix://myserver:1526;InformixServer=myinformixserver;
Database=account;User=jsmith;Password=secret;
```

If successful, the tool will return the time required to connect.

4. At the ISQL> prompt, issue a SQL command to query or modify the data source; then, press **Enter**. For example:

```
SELECT * FROM FEATURES;
```

Note: SQL commands must be terminated by a semi-colon.

Note: In addition to SQL commands, JDBCISQL supports a set of proprietary commands. Type Help at the prompt for a list of supported commands and syntax.

The results of the command are displayed in the terminal.

5. After you are finished executing queries and commands, you can disconnect from the data source by typing the following; then, pressing **Enter**:

```
DISCONNECT;
```

6. Press any key to end the session.

Tableau

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with Tableau. Tableau is a business intelligence software program that allows you to easily create reports and visualized representations of your data. By using the driver with Tableau, you can improve performance when retrieving data while leveraging the driver's relational mapping tools.

To use the driver to access data with Tableau:

1. Navigate to the \lib*xx* subdirectory of the Progress DataDirect installation directory; then, locate the jar file for your driver:

```
informix.jar
```

2. Copy the .jar file for your driver into the following directory:

Windows: C:\Program Files\Tableau\Drivers

Linux: /opt/tableau/tableau_driver/jdbc

3. Open Tableau. From the **Connect** menu, select **Other Databases (JDBC)**.
4. In the **Other Databases (JDBC)** dialog, provide values for the following fields; then, click **Sign In**.
   - **URL**: Copy and paste your connection URL into this field. The following example shows how to connect using the user ID/password authentication.

     ```
     jdbc:datadirect:informix://server4:1526;InformixServer=test1;
     Database=account;
     ```

     Note: See Authentication on page 45 for details.

   - **Dialect**: Select **SQL92** (the default) from the drop-down box.
   - **Username**: If required by the authentication method being used, enter the user name. Alternatively, this value can be specified with the `User` property in the connection string.
   - **Password**: If required by the authentication method being used, enter the password. Alternatively, this value can be specified with the `Password` property in the connection string.
5. The **Data Source** window appears. In the **Schema** field, select the schema for the service you want to use.
6. In the **Table** field, the tables stored in the selected schema are now exposed and available for selection.

You have successfully accessed your data and are now ready to create reports with Tableau. For detailed information, refer to the Tableau product documentation at: https://www.tableau.com/support/help.

DbVisualizer

After you have installed your driver and defined it on the CLASSPATH, you can use the driver to access your data with the third-party DbVisualizer tool. The following topics guide you through using DbVisualizer to add your driver, connect, and execute SQL statements.

Adding a driver

To add a driver with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Tools**>**Driver Manager**. The Driver Manager window opens.
3. From the Driver Manager menu, select **Driver**>**Create Driver**.
4. Click the  button to navigate to the location of the driver jar file; then, click **OK**. The following are the default locations for the driver:

   Windows

   ```
   C:\Program Files\Progress\DataDirect\JDBC\lib\60\informix.jar
   ```

   Linux

   ```
   /opt/Progress/DataDirect/JDBC/lib/60/informix.jar
   ```

5. Provide values for the following fields; then, close the Driver Manager window.

- **Name**: Type an alias for your driver. For example:

  ```
  Informix
  ```

- **URL Format**: Optionally, specify the format of the connection string for your driver. For example:

  ```
  jdbc:datadirect:informix:
  ```

- **Driver Class**: From the drop down menu, select the driver class for your driver. For example:

  ```
  com.ddtek.jdbc.informix.InformixDriver
  ```

You can now use your driver with DbVisualizer. Proceed to "Connecting and executing SQL statements" for information on connecting and executing SQL statements.

Connecting and executing SQL statements

To use the driver to access data with DbVisualizer:

1. Open DbVisualizer.
2. From the menu, select **Database**>**New Connection**. When prompted to use the Connection Wizard, click **OK**.
3. Provide the following information when prompted; then, click **Next** to proceed:
   - **Connection alias**: Type the name to be used when referring to this connection.
   - **Driver**: Select the alias that you provided for your driver from the drop-down menu.
4. Provide values for the following fields; then, click **Finish**.
   - **Database URL**: Copy and paste your connection URL into this field. The following example shows how to connect using the user ID/password authentication.

     ```
     jdbc:datadirect:informix://server4:1526;InformixServer=test1;
     Database=account;User=test;Password=secret;
     ```

     Note: See Authentication on page 45 for details.

5. To execute SQL statements, select **SQL Commander**>**New SQL Commander**. A SQL Commander tab opens.
6. Select values for the following fields:
   - **Database Connection**: Select connection alias you provided for the connection from the drop-down menu.
   - **Schema**: Select the schema you want to execute queries against from the drop-down menu.
7. In the SQL Commander tab, enter SQL commands you want to execute; then select **SQL Commander**>**Execute**. For example:

To select all of the rows from the BLOGS table:

```
SELECT * BLOGS
```

You have successfully accessed your data with DbVisualizer.

3

Configuring and connecting

This section provides information on how to connect to your data store using either the JDBC Driver Manager or DataDirect JDBC data sources, as well as information on how to implement and use functionality supported by the driver.

After the driver has been installed and defined on your classpath, you can connect from your application to your data in either of the following ways.

- Using the JDBC `DriverManager` by specifying the connection URL in the `DriverManager.getConnection()` method.
- Creating a JDBC data source that can be accessed through the Java Naming Directory Interface (JNDI).

For details, see the following topics:

Setting the classpath

The driver must be defined on your CLASSPATH before you can connect. The CLASSPATH is the search string your Java Virtual Machine (JVM) uses to locate JDBC drivers on your computer. If the driver is not defined on your CLASSPATH, you will receive a `class not found` exception when trying to load the driver. Set your system CLASSPATH to include the driver jar file as shown, where *`install_dir`* is the path to your product installation directory.

`install_dir``/lib/60/informix.jar`

Windows Example

```
CLASSPATH=.;C:\Program Files\Progress\DataDirect\JDBC\lib\60\informix.jar
```

UNIX Example

```
CLASSPATH=.:/opt/Progress/DataDirect/JDBC/lib/60/informix.jar
```

Connecting using the JDBC Driver Manager

One way to connect to a service is through the JDBC DriverManager using the `DriverManager.getConnection()` method. As the following example shows, this method specifies a string containing a connection URL.

User ID/password authentication

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:informix://server4:1526;InformixServer=test1;
    Database=account;User=test;Password=secret;");
```

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

Passing the connection URL

After setting the CLASSPATH, the required connection information needs to be passed in the form of a connection URL. The following example includes the properties required for connecting with user ID/password authentication.

Connection URL Syntax

The connection URL takes the following form:

```
jdbc:datadirect:informix://servername:port;InformixServer=informixserver;
Database=databasename;User=userID;Password=password;[property=value[;...]];
```

where:

servername

is the IP address or name of the computer that is running the database server to which you want to connect.

port

is the number of the TCP/IP port.

informixserver

is the name of the database server to which you want to connect.

databasename

is the name of the database to which you want to connect.

userID

specifies the user ID that is used to connect to the Informix database.

password

specifies a password that is used to connect to your Informix database.

property=value

specifies connection property settings. Multiple properties are separated by a semi-colon.

The following example connection string includes the properties required for connecting with the user ID/password authentication.

```
Connection conn = DriverManager.getConnection
  ("jdbc:datadirect:informix://myserver:1526;InformixServer=myinformixserver;
    DatabaseName=account;User=jsmith;Password=secret;");
```

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

See also

Connection property descriptions on page 57
Connection URL examples on page 13
Authentication on page 45

Testing the connection

You can use DataDirect Test™ to verify your connection. The screen shots in this section were taken on a Windows system.

To test the connection from the driver to your data source, follow these steps:

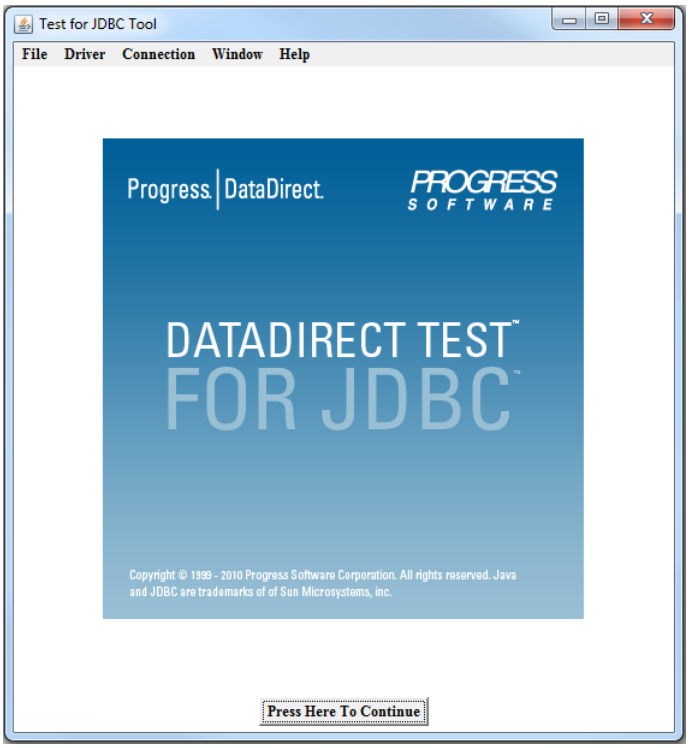
1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`

- UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

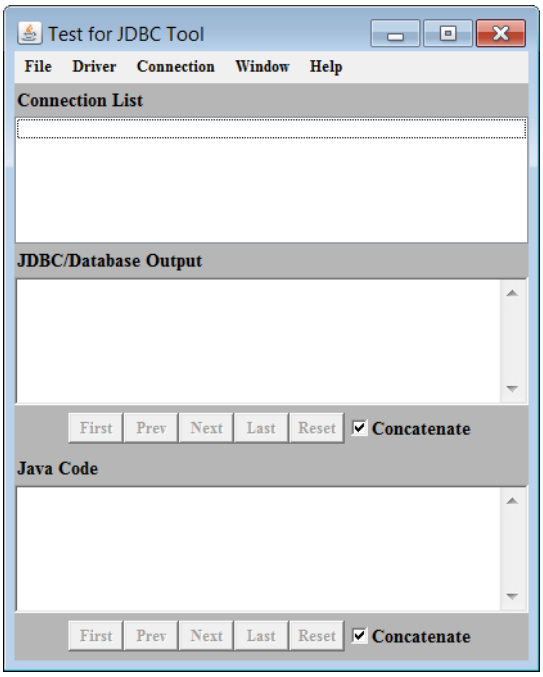
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



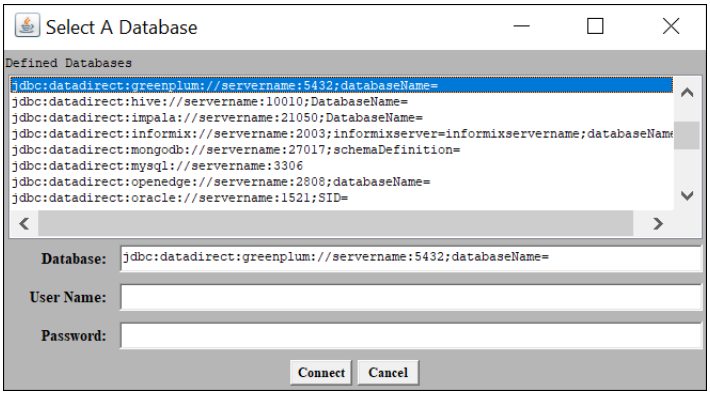
3. Click **Press Here to Continue**.

   The main dialog appears:



4. From the menu bar, select **Connection > Connect to DB**.

   The **Select A Database** dialog appears:



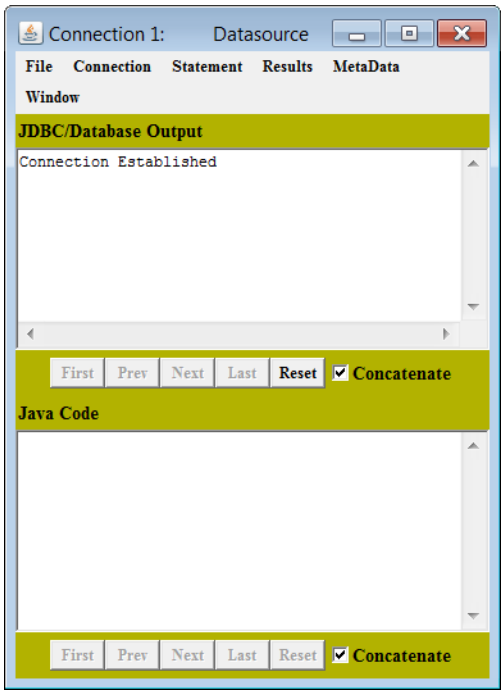
5. Select the appropriate database template from the **Defined Databases** field.
6. In the **Database** field, specify all required connection properties.

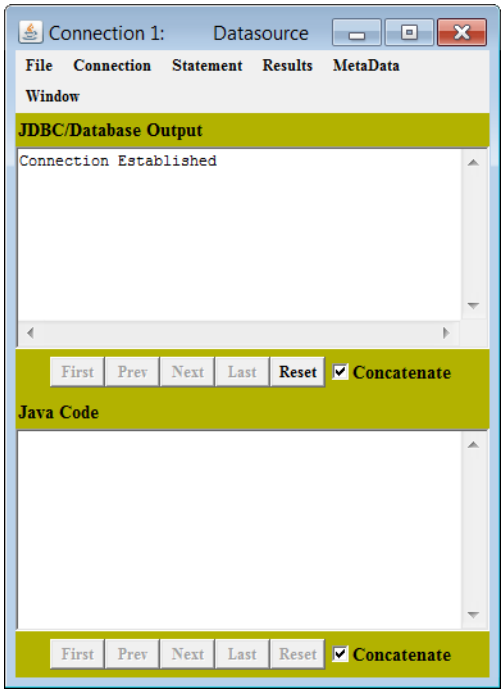
   For example:

   ```
   jdbc:datadirect:informix://server4:1526;InformixServer=test1;
   Database=accounts;
   ```

7. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. (If a connection is not established, the window reports an error.)





Refer to "DataDirect Test" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using DataDirect Test.

Connecting using data sources

A *JDBC data source* is a Java object, specifically a `DataSource` object, that defines connection information required for a JDBC driver to connect to the database. Each JDBC driver vendor provides their own data source implementation for this purpose. A Progress DataDirect data source is Progress DataDirect's implementation of a `DataSource` object that provides the connection information needed for the driver to connect to a database.

Because data sources work with the Java Naming Directory Interface (JNDI) naming service, data sources can be created and managed separately from the applications that use them. Because the connection information is defined outside of the application, the effort to reconfigure your infrastructure when a change is made is minimized. For example, if the database is moved to another database server, the administrator need only change the relevant properties of the `DataSource` object. The applications using the database do not need to change because they only refer to the name of the data source.

How data sources are implemented

Data sources are implemented through a data source class. A data source class implements the following interfaces.

- `javax.sql.DataSource`
- `javax.sql.ConnectionPoolDataSource` (allows applications to use connection pooling)

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

See also

Driver and DataSource classes on page 12

Creating data sources

The following example files provide details on creating and using Progress DataDirect data sources with the Java Naming Directory Interface (JNDI), where *`install_dir`* is the product installation directory.

- *`install_dir`*`/Examples/JNDI/JNDI_LDAP_Example.java` can be used to create a JDBC data source and save it in your LDAP directory using the JNDI Provider for LDAP.
- *`install_dir`*`/Examples/JNDI/JNDI_FILESYSTEM_Example.java` can be used to create a JDBC data source and save it in your local file system using the File System JNDI Provider.

See "Example data source" for an example data source definition for the example files.

To connect using a JNDI data source, the driver needs to access a JNDI data store to persist the data source information. For a JNDI file system implementation, you must download the File System Service Provider from the Oracle Technology Network Java SE Support downloads page, unzip the files to an appropriate location, and add the `fscontext.jar` and `providerutil.jar` files to your CLASSPATH. These steps are not required for LDAP implementations because the LDAP Service Provider is included with supported versions of Java SE.

Example data source

To configure a data source using the example files, you will need to create a data source definition. The content required to create a data source definition is divided into three sections.

First, you will need to import the data source class. For example:

```
import com.ddtek.jdbcx.informix.InformixDataSource;
```

Next, you will need to set the values and define the data source. For example, the following definition contains the minimum properties required to establish connection:

Note:

- Setting the password using a data source is generally not recommended. The data source persists all properties, including the Password property, in clear text.
- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.

```
InformixDataSource mds = new InformixDataSource();
mds.setDescription("My Informix Data Source");
mds.setServerName("myserver");
mds.setPortNumber("1526");
mds.setInformixServer("myinformixserver);
mds.setDatabaseName("payroll");
```

Finally, you will need to configure the example application to print out the data source attributes. Note that this code is specific to the driver and should only be used in the example application. For example, you would add the following section for the minimum properties required to establish a connection:

```
if (ds instance of InformixDataSource)
{
InformixDataSource jmds = (InformixDataSource) ds;
System.out.println("description=" + jmds.getDescription());
System.out.println("serverName=" + jmds.getServerName());
System.out.println("portNumber=" + jmds.getPortNumber());
System.out.println("informixServer=" + jmds.getInformixServer());
System.out.println("database=" + jmds.getDatabase());
System.out.println();
}
```

Calling a data source in an application

Applications can call a Progress DataDirect data source using a logical name to retrieve the `javax.sql.DataSource` object. This object loads the specified driver and can be used to establish a connection to the database.

Once the data source has been registered with JNDI, it can be used by your JDBC application as shown in the following code example.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource)ctx.lookup("EmployeeDB");
Connection con = ds.getConnection("domino", "spark");
```

In this example, the JNDI environment is first initialized. Next, the initial naming context is used to find the logical name of the data source (`EmployeeDB`). The `Context.lookup()` method returns a reference to a Java object, which is narrowed to a `javax.sql.DataSource` object. Then, the `DataSource.getConnection()` method is called to establish a connection.

Testing a data source connection

You can use DataDirect Test™ to establish and test a data source connection. The screen shots in this section were taken on a Windows system.

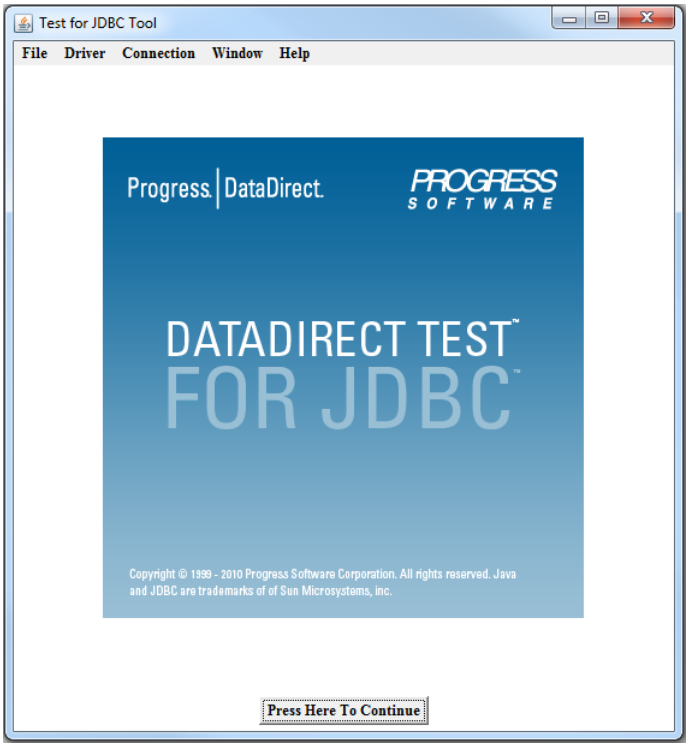
Take the following steps to establish a connection.

1. Navigate to the installation directory. The default location is:
   - Windows systems: `Program Files\Progress\DataDirect\JDBC\testforjdbc`
   - UNIX and Linux systems: `/opt/Progress/DataDirect/JDBC/testforjdbc`

   Note: For UNIX/Linux, if you do not have access to `/opt`, your home directory will be used in its place.

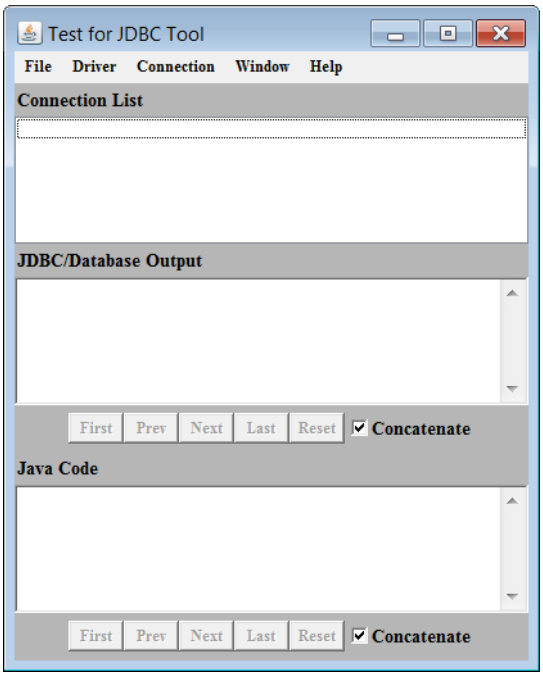
2. From the `testforjdbc` folder, run the platform-specific tool:
   - `testforjdbc.bat` (on Windows systems)
   - `testforjdbc.sh` (on UNIX and Linux systems)

   The **Test for JDBC Tool** window appears:



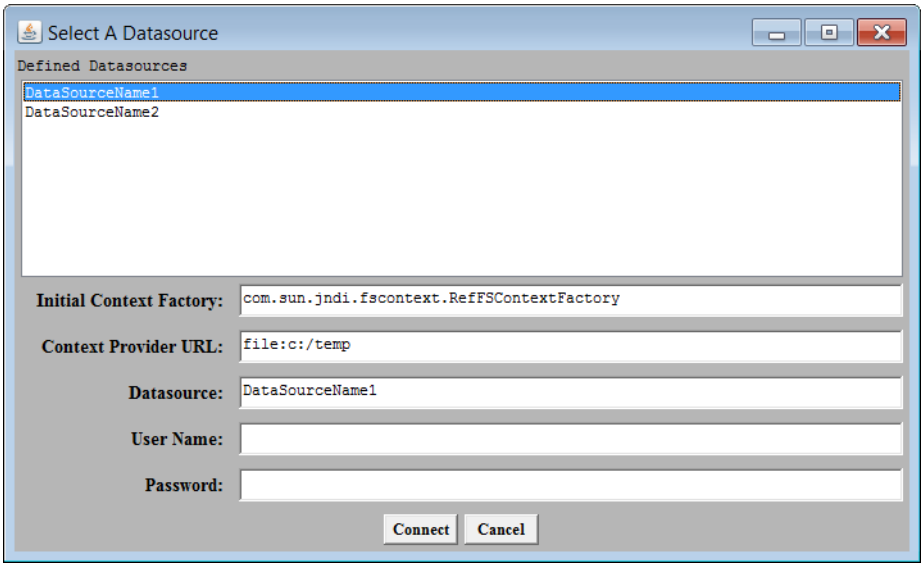
3. Click **Press Here to Continue**.

   The main dialog appears:



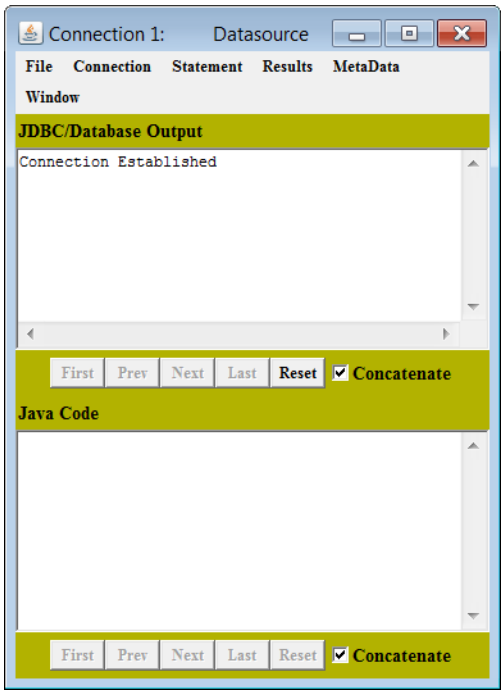
4. From the menu bar, select **Connection > Connect to DB via Data Source**.

   The **Select A Database** dialog appears:



5. Select a datasource template from the **Defined Datasources** field.
6. Provide the following information:
   a) In the **Initial Context Factory**, specify the location of the initial context provider for your application.
   b) In the **Context Provider URL**, specify the location of the context provider for your application.
   c) In the **Datasource** field, specify the name of your datasource.
7. If you are using user ID/password authentication, enter your user ID and password in the corresponding fields.
8. Click **Connect**.

If the connection information is entered correctly, the **JDBC/Database Output** window reports that a connection has been established. If a connection is not established, the window reports an error.



Authentication

The driver supports *User ID/password authentication*. It authenticates the user to the database using a user name and password.

Take the following steps to configure user ID/Password authentication.

- Set the User property to specify the user name that is used to connect to the database.
- Set the Password property to specify the password.
- Set the ServerName property to specify either the IP address or the name of the computer that is running the database server to which you want to connect.
- Set the PortNumber property to specify the TCP port of the primary database server that is listening for connections to the database.
- Set the InformixServer property to specify the name of the database server to which you want to connect.
- Set the DatabaseName property to specify the name of the database to which you want to connect.

The following examples show the connection information required to establish a connection using user ID/password authentication.

Connection URL

```
Connection conn = DriverManager.getConnection
("jdbc:datadirect:informix://myserver:1526;InformixServer=myinformixserver;
  DatabaseName=payroll;User=test;Password=secret);
```

Data Source

```
InformixDataSource mds = new InformixDataSource();
mds.setDescription("My Informix Data Source");
mds.setServerName("myserver");
mds.setPortNumber("1526");
mds.setInformixServer("myinformixserver");
mds.setDatabaseName("payroll");
mds.setUser("jsmith");
mds.setPassword("secret");
```

Note: The User and Password properties are not required to be stored in the connection string. They can also be passed separately by the application.

See also

User on page 91
Password on page 83

Data encryption

All communication between the driver and the server is encrypted using TLS/SSL.

Important: The driver complies with FIPS when FIPS mode is enabled with the client JVM. See "FIPS (Federal Information Processing Standard)" for more information.

FIPS (Federal Information Processing Standard)

The Federal Information Processing Standard (or FIPS) is a cryptography standard created by the U.S. government. FIPS specifications require certain secure algorithms, cryptographic modules, and random number generation. The driver is FIPS compliant for data encryption when FIPS is enabled for the JVM on the client machine.

The following applies when the driver is running in a FIPS environment:

- The driver complies with 140-3 and 140-2 standards.
- The driver uses PKCS #11 providers to access keystores.

The driver was tested with FIPS 140-3 enabled using Red Hat OpenJDK 21 on a Red Hat Universal Base Image 9 instance.

Failover

The driver provides the following levels of failover protection to ensure continuous, uninterrupted access to data.

- *Connection failover* provides failover protection for new connections only. The driver fails over new connections to an alternate, or backup, database server if the primary database server is unavailable, for

example, because of a hardware failure or traffic overload. If a connection to the database is lost, or dropped, the driver does not fail over the connection. This failover method is the default.

- *Extended connection failover* provides failover protection for new connections and lost database connections. If a connection to the database is lost, the driver fails over the connection to an alternate server, preserving the state of the connection at the time it was lost, but not any work in progress.
- *Select Connection failover* provides failover protection for new connections and lost database connections. In addition, it provides protection for Select statements that have work in progress. If a connection to the database is lost, the driver fails over the connection to an alternate server, preserving the state of the connection at the time it was lost and preserving the state of any work being performed by Select statements.

Note: For more information on connection failover and different levels of failover protection provided by the driver, refer to "Failover" in the *Progress DataDirect for JDBC Drivers Reference*.

To configure failover:

1. Specify the primary and alternate servers:
   a. Specify your primary server using a connection URL or data source.
   b. Specify one or multiple alternate servers by setting the AlternateServers property.

      Note: To turn off failover, do not specify a value for the AlternateServers property.

2. Choose a failover method by setting the FailoverMode connection property. The default method is connection failover (`FailoverMode=connect`).
3. If FailoverMode=`extended` or FailoverMode=`select`, set the FailoverGranularity property to specify how you want the driver to behave if exceptions occur while trying to reestablish a lost connection. The default behavior of the driver is to continue with the failover process and post any exceptions on the statement on which they occur (FailoverGranularity=`nonAtomic`).
4. Optionally, configure the connection retry feature by setting the ConnectionRetryCount and ConnectionRetryDelay connection properties.
5. Optionally, set the FailoverPreconnect property if you want the driver to establish a connection with the primary and an alternate server at the same time. The default behavior is to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection (`FailoverPreconnect=false`).

The following examples configure the driver to use connection failover in conjunction with connection retry.

Connection URL

```
jdbc:datadirect:informix://myserver1:2003;InformixServer=myinformixserver1;
DatabaseName=payroll;User=jsmith;Password=secret;
AlternateServers=(myserver2:2003;InformixServer=myinformixserver2,myserver3:2003);
ConnectionRetryCount=2;ConnectionRetryDelay=5
```

In this example:

```
...myserver1:2003;InformixServer=myinformixserver1;DatabaseName=payroll...
```

is the part of the connection URL that specifies connection information for the primary server. Alternate servers are specified using the AlternateServers property. For example:

```
...;AlternateServers=(myserver2:2003;InformixServer=myinformixserver2,myserver3:
2003)
```

If a successful connection is not established on the Informix driver's first pass through the list of database servers (primary and alternate), the driver retries the list of servers in the same sequence twice (`ConnectionRetryCount=2`). Because the connection retry delay has been set to five seconds (`ConnectionRetryDelay=5`), the driver waits five seconds between retry passes.

Data Source

```
InformixDataSource mds = new InformixDataSource();
mds.setDescription("My Informix Data Source");
mds.setServerName("myserver");
mds.setPortNumber("1526");
mds.setInformixServer("myinformixserver1");
mds.setDatabase("payroll");
mds.setUser("jsmith");
mds.setPassword("secret");
mds.setAlternateServers("myserver2:2003;InformixServer=myinformixserver2,
                         myserver3:2003");
```

See also

Connection property descriptions on page 57

Performance considerations

You can optimize application performance by adopting guidelines described in this section.

FetchBufferSize: Decreasing the fetch buffer size reduces memory consumption, but means more network round trips, which decreases performance. Increasing the fetch buffer size improves performance because fewer network round trips are needed to return data from the database.

InsensitiveResultSetBufferSize: To improve performance when using scroll-insensitive result sets, the driver can cache the result set data in memory instead of writing it to disk. By default, the driver caches 2 MB of insensitive result set data in memory and writes any remaining result set data to disk. Performance can be improved by increasing the amount of memory used by the driver before writing data to disk or by forcing the driver to never write insensitive result set data to disk. The maximum cache size setting is 2 GB.

MaxPooledStatements: To improve performance, the driver's own internal prepared statement pooling should be enabled when the driver does not run from within an application server or from within another application that does not provide its own prepared statement pooling. When the driver's internal prepared statement pooling is enabled, the driver caches a certain number of prepared statements created by an application. For example, if the MaxPooledStatements property is set to `20`, the driver caches the last 20 prepared statements created by the application. If the value set for this property is greater than the number of prepared statements used by the application, all prepared statements are cached.

ResultSetMetaDataOptions: The driver's performance may be adversely affected if you set this option to 1. If set to 1 and the ResultSetMetaData.getTableName method is called, the driver performs emulations which take additional processing.

Refer to "Designing JDBC Applications for Performance Optimization" in the *Progress DataDirect for JDBC Drivers Reference* for more information about using prepared statement pooling to optimize performance.

Client information

Many databases allow applications to store client information associated with a connection, which can be useful for database administration and monitoring purposes. The driver allows applications to store and return the following types of client information.

- Name of the application
- User ID
- Host name of the client
- Additional accounting information, such as an accounting ID
- Driver name and version

This information can be used for database administration and monitoring purposes.

Refer to "Client information" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

Bulk load

As Informix does not have native bulk load support, the Informix driver emulates bulk load using the standard batch mechanism.

The BulkLoadBatchSize connection property affects how bulk load works with the Informix driver. It suggests the number of rows to be loaded to the database when bulk loading data. The default is `1000`.

See also

BulkLoadBatchSize on page 65

4

Additional features and functionality

The following section describes additionally supported features and functionality that are specific to the driver.

For details, see the following topics:

- Connection pooling
- Statement pooling
- Isolation levels
- Using scrollable cursors
- JTA support
- Parameter metadata support
- ResultSet metadata support
- Rowset support
- Blob and Clob searches
- Auto-generated keys support

Connection pooling

Progess DataDirect for JDBC drivers support connection pooling using the DataDirect Connection Pool Manager. Typically, creating a connection is the most performance-expensive operations that an application performs. Connection pooling allows you to reuse connections rather than create a new one every time a driver needs to connect to the database. Further, connection pooling manages connection sharing across multiple user requests to maintain performance and reduce the number of new connections to be created.

Refer to "Connection Pool Manager" in the *Progress DataDirect for JDBC Drivers Reference* for more information.

Statement pooling

Most applications have a certain set of SQL statements that are executed multiple times and a few SQL statements that are executed only once or twice during the life of the application. Similar to connection pooling, *statement pooling* provides performance gains for applications that execute the same SQL statements multiple times over the life of the application.

A *statement pool* is a group of prepared statements that can be reused by an application. If you have an application that repeatedly executes the exact same SQL statements, statement pooling can improve performance because the database server does not have to repeatedly parse and create cursors for the same statement. In addition, the associated network round trips to the database server are avoided.

The drivers have an internal prepared statement pooling mechanism, which allows you to realize the performance benefits of statement pooling when you are not running from within an application server or another application that provides its own statement pooling. You can enable this driver-based internal statement pooling with the MaxPooledStatements connection property.

The DataDirect for JDBC drivers also support the DataDirect Statement Pool Monitor. You can use the Statement Pool Monitor to load statements into and remove statements from the statement pool as well as generate information to help you troubleshoot statement pooling performance. The Statement Pool Monitor is an integrated component of the driver, and you can manage statement pooling directly with DataDirect-specific methods. In addition, the Statement Pool Monitor can be enabled as a Java Management Extensions (JMX) MBean. When enabled as a JMX MBean, the Statement Pool Monitor can be used to manage statement pooling with standard JMX API calls, and it can easily be used by JMX-compliant tools, such as JConsole. To enable the Statement Pool Monitor as a JMX MBean, you must register the Statement Pool Monitor MBean with the RegisterStatementPoolMonitorMBean connection property.

Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

See also

MaxPooledStatements on page 82
RegisterStatementPoolMonitorMBean on page 85

Isolation levels

The driver supports the Read Committed, Read Uncommitted, Repeatable Read, and Serializable isolation levels. The default is Read Committed.

Using scrollable cursors

The driver supports scroll-sensitive result sets, scroll-insensitive result sets, and updatable result sets.

Note: When the driver cannot support the requested result set type or concurrency, it automatically downgrades the cursor and generates one or more SQLWarnings with detailed information.

JTA support

JDBC distributed transactions through JTA are supported by the driver.

Parameter metadata support

The driver supports returning parameter metadata.

Insert and Update Statements

The Informix driver supports returning parameter metadata for Insert and Update statements.

Select Statements

The Informix driver supports returning parameter metadata for Select statements that contain parameters in ANSI SQL 92 entry-level predicates, for example, such as COMPARISON, BETWEEN, IN, LIKE, and EXISTS predicate constructs. Refer to the ANSI SQL reference for detailed syntax.

Parameter metadata can be returned for a Select statement if one of the following conditions is true:

- The statement contains a predicate value expression that can be targeted against the source tables in the associated FROM clause. For example:

  ```
  SELECT * FROM foo WHERE bar > ?
  ```

  In this case, the value expression "bar" can be targeted against the table "foo" to determine the appropriate metadata for the parameter.

- The statement contains a predicate value expression part that is a nested query. The nested query's metadata must describe a single column. For example:

  ```
  SELECT * FROM foo WHERE (SELECT x FROM y WHERE z = 1) < ?
  ```

The following Select statements show further examples for which parameter metadata can be returned:

```
SELECT col1, col2 FROM foo WHERE col1 = ? and col2 > ?
SELECT ... WHERE colname = (SELECT col2 FROM t2 WHERE col3 = ?)
SELECT ... WHERE colname LIKE ?
SELECT ... WHERE colname BETWEEN ? and ?
SELECT ... WHERE colname IN (?, ?, ?)
SELECT ... WHERE EXISTS(SELECT ... FROM T2 WHERE col1 < ?)
```

ANSI SQL 92 entry-level predicates in a WHERE clause containing GROUP BY, HAVING, or ORDER BY statements are supported. For example:

```
SELECT * FROM t1 WHERE col = ? ORDER BY 1
```

Joins are supported. For example:

```
SELECT * FROM t1,t2 WHERE t1.col1 = ?
```

Fully qualified names and aliases are supported. For example:

```
SELECT a, b, c, d FROM T1 AS A, T2 AS B WHERE A.a = ? and B.b = ?"
```

When parameter metadata is requested for a column defined as an approximate numeric data type, the driver returns a scale of 255, which indicates the column has an approximate numeric data type and has no scale. For example, suppose we create a table where col2 is an approximate numeric data type with a precision of 20:

```
CREATE table fooTest(col1 int, col2 decimal(20))
```

The driver returns parameter metadata that indicates that col2 has a data type of decimal, a precision of 20, and a scale of 255.

Stored Procedures

The Informix driver does not support returning parameter metadata for stored procedure arguments.

ResultSet metadata support

If your application requires table name information, the driver can return table name information in ResultSet metadata for Select statements. By setting the ResultSetMetaDataOptions property to 1, the driver performs additional processing to determine the correct table name for each column in the result set when the ResultSetMetaData.getTableName() method is called. Otherwise, the getTableNames() method may return an empty string for each column in the result set.

The table name information that is returned by the driver depends on whether the column in a result set maps to a column in a table in the database. For each column in a result set that maps to a column in a table in the database, the driver returns the table name associated with that column. For columns in a result set that do not map to a column in a table (for example, aggregates and literals), the driver returns an empty string.

The Select statements for which ResultSet metadata is returned may contain aliases, joins, and fully qualified names. The following queries are examples of Select statements for which the ResultSetMetaData.getTableName() method returns the correct table name for columns in the Select list:

```
SELECT id, name FROM Employee
SELECT E.id, E.name FROM Employee E
SELECT E.id, E.name AS EmployeeName FROM Employee E
SELECT E.id, E.name, I.location, I.phone FROM Employee E, EmployeeInfo I
   WHERE E.id = I.id
SELECT id, name, location, phone FROM Employee, EmployeeInfo WHERE id = empId
SELECT Employee.id, Employee.name, EmployeeInfo.location, EmployeeInfo.phone
   FROM Employee, EmployeeInfo WHERE Employee.id = EmployeeInfo.id
```

The table name returned by the driver for generated columns is an empty string. The following query is an example of a Select statement that returns a result set that contains a generated column (the column named "upper").

```
SELECT E.id, E.name as EmployeeName, {fn UCASE(E.name)} AS upper FROM Employee E
```

The driver also can return schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information. For example, for the following statement, the driver returns "test" for the catalog name, "test1" for the schema name, and "foo" for the table name:

```
SELECT * FROM test.test1.foo
```

The additional processing required to return table name, schema name, and catalog name information is only performed if the ResultSetMetaData.getTableName(), ResultSetMetaData.getSchemaName(), or ResultSetMetaData.getCatalogName() methods are called.

Rowset support

The driver supports any JSR 114 implementation of the RowSet interface, including:

- CachedRowSets
- FilteredRowSets
- WebRowSets
- JoinRowSets
- JDBCRowSets

Visit https://www.jcp.org/en/jsr/detail?id=114 for more information about JSR 114.

Blob and Clob searches

When searching a Clob value for a string pattern using the Clob.position method, the search pattern must be less than or equal to a maximum value of 4096 bytes. Similarly, when searching a Blob value for a byte pattern using the Blob.position method, the search pattern must be less than or equal to a maximum value of 4096 bytes.

Auto-generated keys support

The driver supports retrieving the values of auto-generated keys. An auto-generated key returned by the driver is the value of a SERIAL column or a SERIAL8 column.

An application can return values of auto-generated keys when it executes an Insert statement. How you return these values depends on whether you are using an Insert statement with a Statement object or with a PreparedStatement object, as outlined in the following scenarios:

- When using an Insert statement with a Statement object, the driver supports the following form of the Statement.execute and Statement.executeUpdate methods to instruct the driver to return values of auto-generated keys:

- `Statement.execute(String sql, int autoGeneratedKeys)`
- `Statement.execute(String sql, int[] columnIndexes)`
- `Statement.execute(String sql, String[] columnNames)`
- `Statement.executeUpdate(String sql, int autoGeneratedKeys)`
- `Statement.executeUpdate(String sql, int[] columnIndexes)`
- `Statement.executeUpdate(String sql, String[] columnNames)`

- When using an Insert statement with a PreparedStatement object, the driver supports the following form of the Connection.prepareStatement method to instruct the driver to return values of auto-generated keys:
  - `Connection.prepareStatement(String sql, int autoGeneratedKeys)`
  - `Connection.prepareStatement(String sql, int[] columnIndexes)`
  - `Connection.prepareStatement(String sql, String[] columnNames)`

  An application can retrieve values of auto-generated keys using the Statement.getGeneratedKeys() method. This method returns a ResultSet object with a column for each auto-generated key.

  Refer to "Designing JDBC applications for performance optimization" in the *Progress DataDirect for JDBC Drivers Reference* for information about how auto-generated keys can improve performance.

5

Connection property descriptions

You can use connection properties to customize the driver for your environment. This section organizes connection properties according to functionality. You can use connection properties with either the JDBC `DriverManager` or a JDBC data source. For a `DriverManager` connection, a property is expressed as a key value pair and takes the form *`property=value`*. For a data source connection, a property is expressed as a JDBC method and takes the form `set`*`property`*`(`*`value`*`)`.

Note:

- In a JDBC data source, string values must be enclosed in double quotation marks, for example, `setUser("abc@defcorp.com")`.
- The data type listed for each connection property is the Java data type used for the property value in a JDBC data source.
- Connection property names are case-insensitive. For example, `Password` is the same as `password`.
- For connection properties that support string values, use the following escape sequence to specify values containing leading or trailing spaces and curly brackets: `{`*`value`*`}`. For example: `User={hello }` or `Password={{hello}}`.

The following tables describe the connection properties by functionality.

- Client information properties
- Statement pooling properties
- Additional properties

General properties

The following table summarizes connection properties required to connect to a database.

Property	Data Source Method	Default
DatabaseName on page 70	`setDatabaseName` `getDatabaseName`	No default value
InformixServer on page 77	`setInformixServer` `getInformixServer`	No default value
PortNumber on page 83	`setPortNumber` `getPortNumber`	No default value
ServerName on page 87	`setServerName` `getServerName`	No default value

User ID/password authentication properties

The following table summarizes the connection properties required for user ID/password authentication.

Property	Data Source Method	Default
User on page 91	`setUser` `getUser`	No default value
Password on page 83	`setPassword` `getPassword`	No default value

Bulk load properties

The following table contains the only connection property that affects how bulk load works with the driver.

Property	Data Source Method	Default
BulkLoadBatchSize on page 65	`setBulkLoadBatchSize` `getBulkLoadBatchSize`	`1000`

Failover properties

The following table summarizes the connection properties used for configuring failover.

Property	Data Source Method	Default
AlternateServers on page 63	setAlternateServers getAlternateServers	No default value
ConnectionRetryCount on page 68	setConnectionRetryCount getConnectionRetryCount	5
ConnectionRetryDelay on page 69	setConnectionRetryDelay getConnectionRetryDelay	1 (second)
FailoverGranularity on page 73	setFailoverGranularity getFailoverGranularity	nonAtomic
FailoverMode on page 73	setFailoverMode getFailoverMode	connect
FailoverPreconnect on page 74	setFailoverPreconnect getFailoverPreconnect	false
LoadBalancing on page 80	setLoadBalancing getLoadBalancing	false

Timeout properties

The following table summarizes timeout connection properties.

Property	Data Source Method	Default
LoginTimeout on page 81	setLoginTimeout getLoginTimeout	0
QueryTimeout on page 85	setQueryTimeout getQueryTimeout	0

Client information properties

The following table summarizes connection properties that can be used to return client information.

Property	Data Source Method	Default
AccountingInfo on page 62	setAccountingInfo getAccountingInfo	No default value

Property	Data Source Method	Default
ApplicationName on page 64	setApplicationName getApplicationName	No default value
ClientHostName on page 66	setClientHostName getClientHostName	No default value
ClientUser on page 67	setClientUser getClientUser	No default value
ProgramID on page 84	setProgramID getProgramID	No default value

Statement pooling properties

The following table summarizes statement pooling connection properties.

Property	Data Source Method	Default
ImportStatementPool on page 76	setImportStatementPool getImportStatementPool	No default value
MaxPooledStatements on page 82	setMaxPooledStatements getMaxPooledStatements	0
RegisterStatementPoolMonitorMBean	setRegisterStatementPoolMonitorMBean getRegisterStatementPoolMonitorMBean	false

Additional properties

The following table summarizes additional connection properties.

Property	Data Source Method	Default
CatalogOptions on page 65	setCatalogOptions getCatalogOptions	2
CodePageOverride on page 67	setCodePageOverride getCodePageOverride	No default value
ConvertNull on page 70	setConvertNull getConvertNull	1

Property	Data Source Method	Default
DBDate on page 71	setDBDate getDBDate	No default value
FetchBufferSize on page 75	setFetchBufferSize getFetchBufferSize	32767
InitializationString on page 77	setInitializationString getInitializationString	No default value
InsensitiveResultSetBufferSize on page 78	setInsensitiveResultSetBufferSize getInsensitiveResultSetBufferSize	2048
JavaDoubleToString on page 79	setJavaDoubleToString getJavaDoubleToString	false
JDBCBehavior on page 80	setJDBCBehavior getJDBCBehavior	1
ResultSetMetaDataOptions on page 86	setResultSetMetaDataOptions getResultSetMetaDataOptions	0
SpyAttributes on page 88	setSpyAttributes getSpyAttributes	No default value
UseDelimitedIdentifier on page 90	setUseDelimitedIdentifier getUseDelimitedIdentifier	true

For details, see the following topics:

- AccountingInfo
- AlternateServers
- ApplicationName
- BulkLoadBatchSize
- CatalogOptions
- ClientHostName
- ClientUser
- CodePageOverride
- ConnectionRetryCount

AccountingInfo

Purpose

Defines accounting information. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the accounting information.

Data Source Methods

```
public String getAccountingInfo()
public void setAccountingInfo(String)
```

Default

No default value

Data Type

String

See also

Client information on page 49

AlternateServers

Purpose

Specifies a list of alternate database servers that is used to failover new or lost connections, depending on the failover method selected. See the FailoverMode property for information about choosing a failover method.

Valid Values

```
(servername1[:port1][;property=value[;...]][,servername2[:port2]
[;property=value[;...]]]...)
```

The server name (*servername1*, *servername2*, and so on) is required for each alternate server entry. Port number (*port1*, *port2*, and so on) and connection properties (*property=value*) are optional for each alternate server entry. If the port is unspecified, the port number of the primary server is used. If the port number of the primary server is unspecified, the default port number of 2003 is used.

Optional connection properties are DatabaseName and InformixServer.

Example

The following URL contains alternate server entries for server2 and server3. The alternate server entries contain the optional InformixServer property.

```
jdbc:datadirect:informix://myserver1:2003;InformixServer=myinformixserver1;
DatabaseName=payroll;AlternateServers=(myserver2:2003;
InformixServer=myinformixserver2,myserver3:2003;InformixServer=myinformixserver3)
```

Data Source Methods

```
public String getAlternateServers()
```

```
public void setAlternateServers(String)
```

Default

No default value

Data type

String

See also

Failover on page 46

ApplicationName

Purpose

Specifies the name of the application. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the name of the application.

Data Source Methods

```
public String getApplicationName()
```

```
public void setApplicationName(String)
```

Default

No default value

Data Type

String

See also

Client information on page 49

BulkLoadBatchSize

Purpose

Provides a suggestion to the driver for the number of rows to load to the database at a time when bulk loading data. Performance can be improved by increasing the number of rows the driver loads at a time because fewer network round trips are required. Be aware that increasing the number of rows that are loaded also causes the driver to consume more memory on the client.

Valid Values

x

where:

x

is a positive integer that represents a number of rows.

Notes

- This property suggests the number of rows regardless of which bulk load method is used: using a DDBulkLoad object or using bulk load for batch inserts.
- The DDBulkObject.setBatchSize() method overrides the value that is set by this property.

  Refer to "JDBC extensions" in the *Progress DataDirect for JDBC Drivers Reference* for more information about bulk load methods.

Data Source Methods

```
public Long getBulkLoadBatchSize()
```

```
public void setBulkLoadBatchSize(Long)
```

Default

`1000`

Data Type

long

See also

Bulk load on page 49

CatalogOptions

Purpose

Determines which type of metadata information is included in result sets when an application calls DatabaseMetaData methods.

Valid Values

2 | 4

Behavior

If set to 2, result sets do not contain synonyms.

If set to 4, a hint is provided to the driver to emulate getColumns() calls using the ResultSetMetaData object instead of querying database catalogs for column information. Result sets contain synonyms. Using emulation can improve performance because the SQL statement that is formulated by the emulation is less complex than the SQL statement that is formulated using getColumns(). The argument to getColumns() must evaluate to a single table. If it does not, because of a wildcard or null value, for example, the driver reverts to the default behavior for getColumns() calls.

Data Source Methods

```
public Integer getCatalogOptions()
```

```
public void setCatalogOptions(Integer)
```

Default

2

Data Type

Integer

ClientHostName

Purpose

Specifies the host name of the client machine. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is the host name of the client machine.

Data Source Methods

```
public String getClientHostName()
```

```
public void setClientHostName(String)
```

Default

No default value

Data Type

String

See also

Client information on page 49

ClientUser

Purpose

Specifies the user ID. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

string

where:

string

is a valid user ID.

Data Source Methods

```
public String getClientUser()
public void setClientUser(String)
```

Default

No default value

Data Type

String

See also

Client information on page 49

CodePageOverride

Purpose

Specifies the name of the code page to be used by the driver to convert Character data. The specified code page overrides the default database code page or column collation. All Character data that is returned from or written to the database is converted using the specified code page.

By default, the driver automatically determines which code page to use to convert Character data. Use this property only if you need to change the driver's default behavior.

Valid Values

string

where:

string

is the name of a valid code page that is supported by your JVM.

Example

CP950

Data Source Methods

```
public String getCodePageOverride()
public void setCodePageOverride(String)
```

Default

No default value

Data Type

String

ConnectionRetryCount

Purpose

Specifies the number of times the driver retries connection attempts to the primary database server, and if specified, alternate servers until a successful connection is established.

Valid Values

0| *x*

where:

x

is a positive integer that represents the number of retries.

Behavior

If set to 0, the driver does not try to reconnect after the initial unsuccessful attempt.

If set to *x*, the driver retries connection attempts the specified number of times. If a connection is not established during the retry attempts, the driver returns an exception that is generated by the last database server to which it tried to connect.

Notes

- If an application sets a login timeout value (for example, using DataSource.loginTimeout or DriverManager.loginTimeout), and the login timeout expires, the driver ceases connection attempts.

- The ConnectionRetryDelay property specifies the wait interval, in seconds, to occur between retry attempts.

Example

If this property is set to 2 and alternate servers are specified using the AlternateServers property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt.

Data Source Methods

```
public Integer getConnectionRetryCount()
```

```
public void setConnectionRetryCount(Integer)
```

Default

5

Data Type

Integer

See also

Failover on page 46

ConnectionRetryDelay

Purpose

Specifies the number of seconds the driver waits between connection retry attempts when ConnectionRetryCount is set to a positive integer.

Valid Values

0| *x*

where:

x

is a number of seconds.

Behavior

If set to 0, the driver does not delay between retries.

If set to *x*, the driver waits between connection retry attempts the specified number of seconds.

Example

If ConnectionRetryCount is set to 2, this property is set to 3, and alternate servers are specified using the AlternateServers property, the driver retries the list of servers (primary and alternate) twice after the initial retry attempt. The driver waits 3 seconds between retry attempts.

Data Source Methods

```
public Integer getConnectionRetryDelay()
```

```
public void setConnectionRetryDelay(Integer)
```

Default

1 (second)

Data Type

Integer

See also

Failover on page 46

ConvertNull

Purpose

Controls how data conversions are handled for null values.

Valid Values

0|1

Behavior

If set to 0, the driver does not perform the data type check if the value of the column is null. This allows null values to be returned even though a conversion between the requested type and the column type is undefined.

If set to 1, the driver checks the data type that is requested against the data type of the table column that stores the data. If a conversion between the requested type and column type is not defined, the driver generates an "unsupported data conversion" exception regardless of the data type of the column value.

Data Source Methods

```
public Integer getConvertNull()
```

```
public void setConvertNull(Integer)
```

Default

1

Data Type

Integer

DatabaseName

Purpose

Specifies the name of the database to which you want to connect.

If this property is not specified, a connection is established to the specified server without connecting to a particular database. A connection that is established to the server without connecting to the database allows an application to use CREATE DATABASE and DROP DATABASE SQL statements. These statements require that the driver cannot be connected to a database. An application can connect to the database after the connection is established by executing the DATABASE SQL statement.

Refer to your IBM Informix documentation for details on using the CREATE DATABASE, DROP DATABASE, and DATABASE SQL statements.

Valid Values

string

where:

string

is the name of an Informix database.

Data Source Methods

```
public String getDatabaseName()
```

```
public void setDatabaseName(String)
```

Default

No default value

Data Type

String

Alias

Database property. If both the Database and DatabaseName properties are specified in a connection URL, the last property that is positioned in the connection URL is used. For example, if your application specifies the following connection URL, the value of the Database connection property would be used instead of the value of the DatabaseName connection property.

```
jdbc:datadirect:informix://myserver1:2003;InformixServer=myinformixserver1;
DatabaseName=jdbc;Database=acct;User=jsmith;Password=secret
```

DBDate

Purpose

Sets the Informix DBDate server option for formatting literal date values when inserting, updating, and retrieving data in DATE columns. Using this property, you can customize the following items:

- Order in which the month, day, and year fields appear in a date string
- Year field to contain two or four digits
- Separator character used to separate the date fields

This property does not affect the format of the string in the date escape syntax. Dates specified using the date escape syntax always use the JDBC escape format: *yyyy-mm-dd*.

Valid Values

```
DMY2[sep_character]  | DMY4[sep_character]  |
MDY2[sep_character]  | MDY4[sep_character]  |
Y4DM[sep_character]  | Y4MD[sep_character]  |
Y2DM[sep_character]
```

where:

D

is a 2-digit day field.

M

is a 2-digit month field.

Y2

is a 2-digit year field.

Y4

is a 4-digit year field.

sep_character

is one of the following characters used to separate the fields: hyphen (-), period (.), or forward slash (/). If a separator is not specified, a forward slash (/) is used to separate the fields.

If unspecified, the format of literal date values conforms to the default server behavior.

Example

A value of `Y4MD-` specifies a date format that has a 4-digit year, followed by the month and then by the day. The date fields are separated by a hyphen (-) (for example: `2004-02-15`).

Data Source Methods

```
public String getDBDate()
```

```
public void setDBDate(String)
```

Default

No default value

Data Type

String

FailoverGranularity

Purpose

Determines how the driver behaves if exceptions occur while trying to reestablish a lost connection. This property is ignored if `FailoverMode=connect`.

Valid Values

`nonAtomic` | `atomic` | `atomicWithRepositioning`

Behavior

If set to `nonAtomic`, the driver continues with the failover process and posts any exceptions on the statement on which they occur.

If set to `atomic`, the driver fails the entire failover process if an exception is generated as the result of restoring the state of the connection. The driver stops trying to connect to an alternative server and returns an exception indicating that the connection was lost. If an exception is generated as a result of restoring the state of work in progress by re-executing the Select statement, the driver continues with the failover process, but generates an exception warning that the Select statement must be reissued.

If set to `atomicWithRepositioning`, the driver fails the entire failover process if any exception is generated as the result of restoring the state of the connection or the state of work in progress. The driver stops trying to connect to an alternative server and returns an exception indicating that the connection was lost.

Data Source Methods

```
public String getFailoverGranularity()
```

```
public void setFailoverGranularity(String)
```

Default

`nonAtomic`

Data Type

String

See also

Failover on page 46

FailoverMode

Purpose

Specifies the type of failover method the driver uses.

Valid Values

`connect` | `extended` | `select`

Behavior

If set to `connect`, the driver provides failover protection for new connections only.

If set to `extended`, the driver provides failover protection for new and lost connections, but not any work in progress.

If set to `select`, the driver provides failover protection for new and lost connections. In addition, it preserves the state of work that is performed by the last Select statement that was executed on the Statement object.

Notes

- The AlternateServers property specifies one or multiple alternate servers for failover and is required for all failover methods. To turn off failover, do not specify a value for the AlternateServers property.
- The FailoverGranularity property determines which action the driver takes if exceptions occur during the failover process.
- The FailoverPreconnect property specifies whether the driver tries to connect to multiple database servers (primary and alternate) at the same time.

Data Source Methods

```
public String getFailoverMode()
```

```
public void setFailoverMode(String)
```

Default

`connect`

Data Type

String

See also

Failover on page 46

FailoverPreconnect

Purpose

Specifies whether the driver tries to connect to the primary and an alternate server at the same time. This property is ignored if `FailoverMode=connect`.

Valid Values

`true | false`

Behavior

If set to `true`, the driver tries to connect to the primary and an alternate server at the same time. This can be useful if your application is time-sensitive and cannot absorb the wait for the failover connection to succeed.

If set to `false`, the driver tries to connect to an alternate server only when failover is caused by an unsuccessful connection attempt or a lost connection. This value provides the best performance, but your application typically experiences a short wait while the failover connection is attempted.

Notes

- The AlternateServers property specifies one or multiple alternate servers for failover.

Data Source Methods

```
public Boolean getFailoverPreconnect()
public void setFailoverPreconnect(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Failover on page 46

FetchBufferSize

Purpose

Specifies the size (in bytes) of the fetch buffer that the driver uses when retrieving data from the database.

Decreasing the fetch buffer size reduces memory consumption, but means more network round trips, which decreases performance. Increasing the fetch buffer size improves performance because fewer network round trips are needed to return data from the database.

To determine the optimal value, use the following formula:

```
X = A * B * 50
```

where:

A

is the number of rows that your application returns when executing Select statements

B

is the number of row columns that are typically returned when executing Select statements.

Valid Values

x

where:

x

is a positive integer from `1` to `32767` that represents the size of the fetch buffer.

Data Source Methods

```
public Integer getFetchBufferSize()
public void setFetchBufferSize(Integer)
```

Default

32767

Data Type

Integer

See also

Performance considerations on page 48

ImportStatementPool

Purpose

Specifies the path and file name of the file to be used to load the contents of the statement pool. When this property is specified, statements are imported into the statement pool from the specified file.

If the driver cannot locate the specified file when establishing the connection, the connection fails and the driver throws an exception.

Valid Values

string

where:

`string`

is the path and file name of the file to be used to load the contents of the statement pool.

Data Source Methods

```
public String getImportStatementPool()
public void setImportStatementPool(String)
```

Default

No default value

Data Type

String

See also

- Statement pooling on page 52
- Performance considerations on page 48

- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

InformixServer

Purpose

Specifies the name of the Informix database server to which you want to connect.

Valid Values

string

where:

string

is the name of the Informix database server.

Data Source Methods

```
public String getInformixServer()
public void setInformixServer(String)
```

Default

No default value

Data Type

String

InitializationString

Purpose

Specifies one or multiple SQL commands to be executed by the driver after it has established the connection to the database and has performed all initialization for the connection. If the execution of a SQL command fails, the connection attempt also fails and the driver throws an exception indicating which SQL command or commands failed.

Valid Values

command [[; *command*]...]

where:

command

is a SQL command.

Notes

- Multiple commands must be separated by semicolons. In addition, if this property is specified in a connection URL, the entire value must be enclosed in parentheses when multiple commands are specified.

Example

```
jdbc:datadirect:informix://myserver1:2003;InformixServer=myinformixserver;
DatabaseName=Test;InitializationString=(command1;command2)
```

Data Source Methods

```
public String getInitializationString()
```

```
public void setInitializationString(String)
```

Default

No default value

Data Type

String

InsensitiveResultSetBufferSize

Purpose

Determines the amount of memory used by the driver to cache insensitive result set data.

Valid Values

`-1` | `0` | *x*

where:

x

is a positive integer that represents the size of the memory buffer.

Behavior

If set to `-1`, the driver caches insensitive result set data in memory. If the size of the result set exceeds available memory, an OutOfMemoryException is generated. With no need to write result set data to disk, the driver processes the data efficiently.

If set to `0`, the driver caches insensitive result set data in memory, up to a maximum of 2 GB. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. Because result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk.

If set to *x*, the driver caches insensitive result set data in memory and uses this value to set the size (in KB) of the memory buffer for caching insensitive result set data. If the size of the result set data exceeds available memory, the driver pages the result set data to disk. Because the result set data may be written to disk, the driver may have to reformat the data to write it correctly to disk. Specifying a buffer size that is a power of 2 results in efficient memory use.

Data Source Methods

```
public Integer getInsensitiveResultSetBufferSize()
```

```
public void setInsensitiveResultSetBufferSize(Integer)
```

Default

2048

Data Type

Integer

See also

Performance considerations on page 48

JavaDoubleToString

Purpose

Determines which algorithm the driver uses when converting a double or float value to a string value. By default, the driver uses its own internal conversion algorithm, which improves performance.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver uses the JVM algorithm when converting a double or float value to a string value. If your application cannot accept rounding differences and you are willing to sacrifice performance, set this value to `true` to use the JVM conversion algorithm.

If set to `false`, the driver uses its own internal algorithm when converting a double or float value to a string value. This value improves performance, but slight rounding differences within the allowable error of the double and float data types can occur when compared to the same conversion using the JVM algorithm.

Data Source Methods

```
public Boolean getJavaDoubleToString()
```

```
public void setJavaDoubleToString(Boolean)
```

Default

`false`

Data Type

Boolean

JDBCBehavior

Purpose

Determines how the driver describes database data types that map to the following JDBC 4.0 data types: NCHAR, NVARCHAR, NLONGVARCHAR, NCLOB, and SQLXML.

Valid Values

`0 | 1`

Behavior

If set to `0`, the driver describes the data types as JDBC 4.0 data types.

If set to `1`, the driver describes the data types using JDBC 3.0-equivalent data types. This allows your application to continue using JDBC 3.0 types.

Data Source Methods

```
public Integer getJDBCBehavior()
```

```
public void setJDBCBehavior(Integer)
```

Default

`1`

Data Type

Integer

LoadBalancing

Purpose

Determines whether the driver uses client load balancing in its attempts to connect to the database servers (primary and alternate). You can specify one or multiple alternate servers by setting the AlternateServers property.

Valid Values

`true | false`

Behavior

If set to `true`, the driver uses client load balancing and attempts to connect to the database servers (primary and alternate) in random order. The driver randomly selects from the list of primary and alternate servers which server to connect to first. If that connection fails, the driver again randomly selects from this list of servers until all servers in the list have been tried or a connection is successfully established.

If set to `false`, the driver does not use client load balancing and connects to each server based on their sequential order (primary server first, then, alternate servers in the order they are specified).

Data Source Methods

```
public Boolean getLoadBalancing()
```

```
public void setLoadBalancing(Boolean)
```

Default

`false`

Data Type

Boolean

See also

Failover on page 46

LoginTimeout

Purpose

Specifies the amount of time, in seconds, that the driver waits for a connection to be established before timing out the connection request.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents a number of seconds.

Behavior

If set to `0`, the driver does not time out a connection request.

If set to *x*, the driver waits for the specified number of seconds before returning control to the application and throwing a timeout exception.

Data Source Methods

```
public Integer getLoginTimeout()
```

```
public void setLoginTimeout(Integer)
```

Default

`0`

Data Type

Integer

MaxPooledStatements

Purpose

Specifies the maximum number of prepared statements to be pooled for each connection and enables the driver's internal prepared statement pooling when set to an integer greater than zero (0). The driver's internal prepared statement pooling provides performance benefits when the driver is not running from within an application server or another application that provides its own statement pooling.

Valid Values

`0` | *x*

where:

x

is a positive integer that represents a number of prepared statements to be cached.

Behavior

If set to `0`, the driver's internal prepared statement pooling is not enabled.

If set to *x*, the driver's internal prepared statement pooling is enabled and the driver uses the specified value to cache up to that many prepared statements created by an application. If the value set for this property is greater than the number of prepared statements that are used by the application, all prepared statements that are created by the application are cached. Because CallableStatement is a sub-class of PreparedStatement, CallableStatements also are cached.

Notes

When you enable statement pooling, your applications can access the Statement Pool Monitor directly with DataDirect-specific methods. However, you can also enable the Statement Pool Monitor as a JMX MBean. To enable the Statement Pool Monitor as an MBean, statement pooling must be enabled with MaxPooledStatements and the Statement Pool Monitor MBean must be registered using the RegisterStatementPoolMonitorMBean connection property.

Example

If the value of this property is set to `20`, the driver caches the last 20 prepared statements that are created by the application.

Data Source Methods

```
public Integer getMaxPooledStatements()
```

```
public void setMaxPooledStatements(Integer)
```

Default

`0`

Data Type

Integer

See also

- Statement pooling on page 52
- Performance considerations on page 48
- RegisterStatementPoolMonitorMBean on page 85
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

Password

Purpose

Specifies a password that is used to connect to your Informix database. A password is required if security is enabled on your database. Contact your system administrator to obtain your password.

Valid Values

string

where:

string

is a valid password. The password is case-sensitive.

Data Source Methods

```
public String getPassword()
```

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

Default

No default value

Data Type

String

PortNumber

Purpose

Specifies the TCP port of the primary database server that is listening for connections to the Informix database.

This property is supported only for data source connections.

Valid Values

port

where:

port

is the port number.

Data Source Methods

```
public Integer getPortNumber()
public void setPortNumber(Integer)
```

Default

Varies depending on operating system

Data Type

Integer

ProgramID

Purpose

The driver and version information on the client to be stored in the database. This value is stored locally and is used for database administration/monitoring purposes.

Valid Values

```
string
```

where:

string

is a value that identifies the product and version of the driver on the client.

Example

```
DDJ04200
```

Data Source Methods

```
public String getProgramID()
public void setProgramID(String)
```

Default

No default value

Data Type

String

See also

Client information on page 49

QueryTimeout

Purpose

Sets the default query timeout (in seconds) for all statements created by a connection.

Valid Values

`-1` | `0` | *x*

where:

x

is the number of seconds.

Behavior

If set to `-1`, the query timeout functionality is disabled. The driver silently ignores calls to the Statement.setQueryTimeout() method.

If set to `0`, the default query timeout is infinite (the query does not time out).

If set to *x*, the driver uses the value as the default timeout for any statement created by the connection. To override the default timeout value that is set by this connection option, call the Statement.setQueryTimeout() method to set a timeout value for a particular statement.

Data Source Methods

```
public Integer getQueryTimeout()
```

```
public void setQueryTimeout(Integer)
```

Default

`0`

Data Type

Integer

RegisterStatementPoolMonitorMBean

Purpose

Registers the Statement Pool Monitor as a JMX MBean when statement pooling has been enabled with MaxPooledStatements. This allows you to manage statement pooling with standard JMX API calls and to use JMX-compliant tools, such as JConsole.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver registers an MBean for the statement pool monitor for each statement pool. This gives applications access to the Statement Pool Monitor through JMX when statement pooling is enabled.

If set to `false`, the driver does not register an MBean for the statement pool monitor for any statement pool.

Notes

Registering the MBean exports a reference to the Statement Pool Monitor. The exported reference can prevent garbage collection on connections if the connections are not properly closed. When garbage collection does not take place on these connections, out of memory errors can occur.

Data Source Methods

```
public Boolean getRegisterStatementPoolMonitorMBean()
```

```
public void setRegisterStatementPoolMonitorMBean(Boolean)
```

Default

`false`

Data Type

Boolean

See also

- Statement pooling on page 52
- MaxPooledStatements on page 82
- Refer to "Statement Pool Monitor" in the *Progress DataDirect for JDBC Drivers Reference* for further details.

ResultSetMetaDataOptions

Purpose

Determines whether the driver returns table name information in the ResultSet metadata for Select statements.

Valid Values

`0 | 1`

Behavior

If set to `0` and the ResultSetMetaData.getTableName() method is called, the driver does not perform additional processing to determine the correct table name for each column in the result set. The getTableName() method may return an empty string for each column in the result set.

If set to `1` and the ResultSetMetaData.getTableName() method is called, the driver performs additional processing to determine the correct table name for each column in the result set. The driver returns schema name and catalog name information when the ResultSetMetaData.getSchemaName() and ResultSetMetaData.getCatalogName() methods are called if the driver can determine that information.

Data Source Methods

```
public Integer getResultSetMetaDataOptions()
```

```
public void setResultSetMetaDataOptions(Integer)
```

Default

`0`

Data Type

Integer

See also

Performance considerations on page 48

ServerName

Purpose

Specifies either the IP address in IPv4 or IPv6 format, or the name of the computer that is running the database server to which you want to connect.

This property is supported only for data source connections.

Valid Values

`string`

where:

`string`

is a valid IP address or server name.

Example

`122.23.15.12` or `InformixServer`

Data Source Methods

```
public String getServerName()
```

```
public void setServerName(String)
```

Default

No default value

Data Type

String

SpyAttributes

Purpose

Enables DataDirect Spy to log detailed information about calls issued by the driver on behalf of the application. DataDirect Spy is not enabled by default.

Valid Values

(*spy_attribute* [; *spy_attribute*]...)

where:

spy_attribute

is any valid DataDirect Spy attribute.

Behavior

Attribute	Description
linelimit=*numberofchars*	Sets the maximum number of characters that DataDirect Spy logs on a single line. The default is 0 (no maximum limit).
log=(file)*filename*	Directs logging to the file specified by *filename*. For Windows, if coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(file)C:\\temp\\spy.log;logIS=yes;logTName=yes.

Attribute	Description
log=(filePrefix)*file_prefix*	Directs logging to a file prefixed by *file_prefix*. The log file is named *file_prefixX*.log where: *X* is an integer that increments by 1 for each connection on which the prefix is specified. For example, if the attribute log=(filePrefix)C:\\temp\\spy_ is specified on multiple connections, the following logs are created: C:\temp\spy_1.log C:\temp\spy_2.log C:\temp\spy_3.log ... If coding a path to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: log=(filePrefix)C:\\temp\\spy_;logIS=yes;logTName=yes.
log=System.out	Directs logging to the Java output standard, System.out.
logIS={ yes | no | nosingleread }	Specifies whether DataDirect Spy logs activity on InputStream and Reader objects. When logIS=nosingleread, logging on InputStream and Reader objects is active; however, logging of the single-byte read InputStream.read or single-character Reader.read is suppressed to prevent generating large log files that contain single-byte or single character read messages. The default is no.
logLobs={ yes | no }	Specifies whether DataDirect Spy logs activity on BLOB and CLOB objects.
logTName={ yes | no }	Specifies whether DataDirect Spy logs the name of the current thread. The default is no.
timestamp={ yes | no }	Specifies whether a timestamp is included on each line of the DataDirect Spy log. The default is yes.

Notes

- If coding a path on Windows to the log file in a Java string, the backslash character (\) must be preceded by the Java escape character, a backslash. For example: `log=(file)C:\\temp\\spy.log`.
- If a log file name does not include the `.log` extension, the driver automatically appends it. For example, a file named `spy.jsp` is renamed to `spy.jsp.log` by the driver.
- For more information, refer to "Tracking JDBC calls with DataDirect Spy" in the *Progress DataDirect for JDBC Drivers Reference*.

Example

The following value instructs the driver to log all JDBC activity to a file using a maximum of 80 characters for each line.

```
(log=(file)/tmp/spy.log;linelimit=80)
```

Data Source Methods

```
public String getSpyAttributes()
```

```
public void setSpyAttributes(String)
```

Default

No default value

Data Type

String

UseDelimitedIdentifier

Purpose

Controls how the Informix server interprets double quote (") characters in SQL statements.

Valid Values

`true` | `false`

Behavior

If set to `true`, the driver sets the Informix DELIMIDENT server option, causing the Informix server to interpret strings enclosed in double quotes as identifiers, not as string literals.

If set to `false`, the driver does not set the Informix DELIMIDENT server option and the Informix server interprets strings enclosed in double quotes as string literals, not as identifiers.

Notes

- If the DELIMIDENT environment variable is set on the server, the driver cannot change the setting. In this case, any value set by this property is ignored.

Data Source Methods

```
public Boolean getUseDelimitedIdentifier()
public void setUseDelimitedIdentifier(Boolean)
```

Default

`true`

Data Type

Boolean

User

Purpose

Specifies the user name that is used to connect to the Informix database.

Valid Values

string

where:

string

is a valid user name. The user name is case-insensitive.

Data Source Methods

```
public String getUser()
public void setUser(String)
```

Default

No default value

Data Type

String