opentext™

OpenText™ Identity Manager
Driver for JDBC Implementation Guide

January 2025

Legal Notice

Copyright 2009 - 2024 Open Text.

The only warranties for products and services of Open Text and its affiliates and licensors (“Open Text”) are as may be set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Open Text shall not be liable for technical or editorial errors or omissions contained herein. The information contained herein is subject to change without notice.

Contents

About this Book and the Library **11**

1 Understanding the OpenText Identity Manager Driver for JDBC **13**

Components for Data Synchronization . . . 13
JDBC . . . 13
Third-Party JDBC Driver . . . 14
Identity Vault . . . 14
Schema . . . 14
Logical Database Class . . . 15
XDS . . . 15
Database Concepts . . . 16
Structured Query Language . . . 16
Data Manipulation Language . . . 16
Data Definition Language . . . 16
View . . . 17
Identity Columns/Sequences . . . 17
Transaction . . . 18
Stored Procedures or Functions . . . 18
Trigger . . . 19
Instead-Of-Trigger . . . 20
How the Driver Works . . . 21
Generic JDBC Driver . . . 21
Supported Operations . . . 21
Supported Databases . . . 21
Supported Third Party JDBC Drivers . . . 21
Support for Password Synchronization . . . 21
Supported Data Synchronization Models . . . 21
Triggerless vs. Triggered Publication . . . 24
Planning to Install the Driver . . . 25
Installation Requirements . . . 25
Options for Installing the Driver Shim . . . 25

2 Installing the Driver Files **27**

Installing the Driver Files . . . 27
Installing JDBC Driver Jar Files . . . 27

3 The Association Utility **29**

Independent Operations . . . 29
Before You Begin . . . 30
Using the Association Utility . . . 31
Parameters for Searching and Replacing . . . 32

4 Installing and Configuring Database Objects 33

SQL Script Conventions . . . 33
Installing IBM DB2 Universal Database (UDB) . . . 35
Installing Informix Dynamic Server (IDS) . . . 36
Installing Microsoft SQL Server . . . 36
Installing MySQL . . . 37
Installing Oracle . . . 37
Installing PostgreSQL 8 . . . 37
Installing PostgreSQL 9 . . . 38
Installing Azure Database for PostgreSQL 14 . . . 39
Installing Sybase Adaptive Server Enterprise (ASE) . . . 40
Testing the Database Object Installation . . . 40

5 Creating a New Driver Object 43

Creating the Driver Object in Designer . . . 43
- Importing the Current Driver Packages . . . 43
- Installing the Driver Packages . . . 44
- Configuring the Driver Object . . . 48
- Deploying the Driver Object . . . 48
- Starting the Driver Object . . . 49

Activating the Driver . . . 49
JDBC Driver Settings . . . 50

6 Configuring the JDBC Driver 53

Smart Configuration . . . 53
- Specifying Custom Descriptor Files . . . 54
- Reserved Filenames for Descriptor Files . . . 54
- Import Descriptor Files . . . 54
- Descriptor File Locations . . . 54
- Precedence . . . 55
- Custom Descriptor Best Practices . . . 55
- Descriptor File DTDs . . . 55

Configuration Parameters . . . 56
- Viewing Driver Parameters . . . 56
- Deprecated Parameters . . . 56
- Authentication Parameters . . . 57

Driver Parameters . . . 57
- Uncategorized Parameters . . . 59
- Database Scoping Parameters . . . 64
- Connectivity Parameters . . . 68
- Compatibility Parameters . . . 71

Subscription Parameters . . . 82
- Uncategorized Parameters . . . 83
- Primary Key Parameters . . . 85

Publication Parameters . . . 90
- Uncategorized Parameters . . . 91
- Triggered Publication Parameters . . . 94
- Triggerless Publication Parameters . . . 96
- Polling Parameters . . . 97

Trace Levels . . . 100

Configuring Third-Party JDBC Drivers 101
Configuring jTDS Support for the JDBC Driver 101

7 Upgrading an Existing Driver 103

What's New 103
What's New in Version 4.1.0.1 103
What's New in Version 4.1.0 103
Working with MapDB 3.0.5 103
Understanding Identity Manager 4.7 Engine Support for Driver Versions 104
Manually Removing the MapDB Cache Files 104
Considerations for Upgrading the Driver With Different Identity Manager and MapDB Versions 104
Upgrading the Driver 105
. . . . . Upgrading the Installed Packages 106
Updating the Driver Files 107

8 Managing the Driver 109

9 Schema Mapping 111

High-Level View 111
Logical Database Classes 111
Indirect Synchronization 111
Mapping OpenText eDirectory Classes to Logical Database Classes 112
Parent Tables 114
Parent Table Columns 115
Child Tables 115
Referential Attributes 117
Single-Value Referential Attributes 117
Multivalue Referential Attributes 118
Direct Synchronization 120
View Column Meta-Identifiers 121
Primary Key Columns 123
Schema Mapping 123
Synchronizing Primary Key Columns 123
Synchronizing Multiple Classes 124
Mapping Multivalue Attributes to Single-Value Database Fields 124

10 Mapping XDS Events to SQL Statements 127

Mapping XDS Events for Indirect Synchronization 127
Mapping XDS Events for Direct Synchronization 127

11 The Event Log Table 129

Event Log Columns 129
record_id 129
table_key 129
status 130
event_type 130

event_time . . . 131
perpetrator . . . 131
table_name . . . 131
column_name . . . 131
old_value . . . 131
new_value . . . 132
Event Types . . . 132

12 Embedded SQL Statements in XDS Events 141

Common Uses of Embedded SQL . . . 142
Embedded SQL Basics . . . 142
Elements . . . 142
Namespaces . . . 142
Embedded SQL Example . . . 143
Token Substitution . . . 143
Virtual Triggers . . . 146
Manual vs. Automatic Transactions . . . 147
Transaction Isolation Level . . . 148
Statement Type . . . 149
SQL Queries . . . 150
Data Definition Language (DDL) Statements . . . 152
Logical Operations . . . 153
Implementing Password Set with Embedded SQL . . . 154
Implementing Modify Password with Embedded SQL . . . 155
Implementing Check Object Password . . . 155
Calling Stored Procedures and Functions . . . 156
Using Embedded SQL to Call Stored Procedures or Functions . . . 156
Using the jdbc:call-procedure Element . . . 157
Using the jdbc:call-function Element . . . 161
Best Practices . . . 165

13 Supported Databases 167

Database Interoperability . . . 167
Supported Databases . . . 167
Database Characteristics . . . 168
Database Features . . . 168
Current Time Stamp Statements . . . 169
Syntaxes for Calling Stored Procedures and Functions . . . 170
Left Outer Join Operators . . . 171
Undelimited Identifier Case Sensitivity . . . 171
Supported Transaction Isolation Levels . . . 172
Commit Keywords . . . 172
IBM DB2 Universal Database (UDB) . . . 173
Informix Dynamic Server (IDS) . . . 174
Microsoft SQL Server . . . 175
MySQL/MariaDB . . . 175
Oracle . . . 176
PostgreSQL . . . 177
Sybase Adaptive Server Enterprise (ASE) . . . 178

14 Third-Party JDBC Drivers **181**

Third-Party JDBC Driver Interoperability . . . 181
Third-Party JDBC Driver Types . . . 181
Driver Types . . . 182
Which Type To Use? . . . 182
Supported Third-Party JDBC Drivers (Recommended) . . . 182
Third-Party JDBC Driver Features . . . 183
JDBC URL Syntaxes . . . 183
JDBC Driver Class Names . . . 184
Supported Third-Party Jar File Placement . . . 185
IBM DB2 Universal Database Type 4 JDBC Driver . . . 185
Informix JDBC Driver . . . 186
Microsoft JDBC Driver for SQL Server . . . 188
jTDS JDBC Driver . . . 189
MySQL Connector/J JDBC Driver . . . 191
Oracle Thin Client JDBC Driver . . . 192
Oracle OCI JDBC Driver . . . 194
PostgreSQL JDBC Driver . . . 195
Sybase Adaptive Server Enterprise JConnect JDBC Driver . . . 196
MariaDB Connector/J JDBC Driver . . . 197
Supported Third-Party JDBC Drivers (Not Recommended) . . . 198
Third-Party JDBC Driver Features . . . 198
JDBC URL Syntaxes . . . 198
JDBC Driver Class Names . . . 199
IBM DB2 Universal Database JDBC Driver . . . 199
Microsoft SQL Server 2008 JDBC Driver . . . 200
Microsoft SQL Server 2008 R2 JDBC Driver . . . 201
Deprecated Third-Party JDBC Drivers . . . 202
Other Third-Party JDBC Drivers . . . 203
IBM Toolbox for Java/JTOpen . . . 203
Minimum Third-Party JDBC Driver Requirements . . . 203
Considerations When Using Other Third-Party JDBC Drivers . . . 204
Security Issues . . . 204

15 Troubleshooting the JDBC Driver **205**

DirXML-Accounts Attribute Shows Incorrect Value When a User is Enabled or Disabled in the OpenText Identity Vault on DB2 and Oracle Database Drivers . . . 205
Password Changes for Users Are Not Synchronized from the OpenText Identity Vault for the Oracle Database Driver . . . 205
Modifying the Default Configuration for the Sybase Driver in Direct Mode . . . 206
Driver Shim Is Irresponsive When a Connected Database Server Does Not Respond . . . 206
Recognizing Publication Events . . . 206
Executing Test Scripts . . . 207
Applying the Latest Driver Package Does Not Change the Default Setting of Enable Service Channel ECV . . . 207
Troubleshooting Driver Processes . . . 207
Null Pointer Exception Appears in the Trace for JDBC Driver for Connecting to Oracle Database While Using ojdbc10.jar . . . 208

A Uninstalling the Driver **209**

Deleting Identity Manager Driver Objects 209
Running the Product Uninstaller 209
Executing Database Uninstallation Scripts 209
IBM DB2 Universal Database (UDB) Uninstallation. 210
Informix Dynamic Server (IDS) Uninstallation 210
Microsoft SQL Server Uninstallation 210
MySQL Uninstallation 211
Oracle Uninstallation 211
PostgreSQL Uninstallation. 211
Sybase Adaptive Server Enterprise (ASE) Uninstallation 212

B Known Issues and Limitations **213**

Known Issues 213
Limitations 214

C Best Practices **215**

Tips for Synchronizing Millions of User Records on the Publisher Channel 215
Schema Name Use Cases 215

D FAQ **221**

Can’t See Tables or Views. 221
Synchronizing with Tables 221
Processing Rows in the Event Log Table 222
Managing Database User Accounts. 222
Synchronizing Large Data Types. 222
Slow Publication 222
Synchronizing Multiple Classes 223
Encrypted Transport 223
Mapping Multivalue Attributes 223
Synchronizing Garbage Strings. 223
Running Multiple JDBC Driver Instances 224

E Supported Data Types **225**

F java.sql.DatabaseMetaData Methods **227**

G JDBC Interface Methods **229**

H Third-Party JDBC Driver Descriptor DTD **235**

I Third-Party JDBC Driver Descriptor Import DTD **237**

J Database Descriptor DTD **239**

K Database Descriptor Import DTD **241**

L Policy Example: Trigger-less Future Event Processing **243**

M Setting Up an OCI Client on Linux **245**

Downloading the Instant Client 245
Setting Up the OCI Client 245
Configuring the OCI Driver 246

N Sybase Chain Modes and the OpenText Identity Manager JDBC driver **247**

Error Codes 247
Procedures and Modes 248
Using Stored Procedure sp_proxmode 248
Chained and Unchained Modes 249
Managing Transactions in a Policy 249
Useful Links 250

O Driver Properties **251**

Driver Configuration 251
Driver Module 251
Authentication 252
Startup Option 252
Driver Parameters 253
ECMAScript 255
Global Configuration 255
Global Configuration Values 255
Global Configuration Values 256
Managed System Information 256
Entitlements 258
Account Tracking 260
Password Synchronization 260
JDBC Fanout Common 261

About this Book and the Library

The OpenText *Identity Manager Driver for Java Database Connectivity (JDBC) Implementation Guide* provides a generic solution for synchronizing data between an OpenText Identity Vault and relational databases. This guide provides an overview of the driver's technology as well as configuration instructions.

Intended Audience

This book provides information for individuals responsible for understanding administration concepts and implementing a secure, distributed administration model.

Other Information in the Library

For more information about the library for OpenText Identity Manager, see the following resources:

- OpenText™ Identity Manager CE 24.4 (v4.10) documentation website (https://www.netiq.com/documentation/identity-manager-4.10/)
- OpenText™ Identity Manager CE 24.4 (v4.10) driver documentation website (https://www.netiq.com/documentation/identity-manager-4.10-drivers/)

1 Understanding the OpenText Identity Manager Driver for JDBC

The OpenText Identity Manager Driver for Java DataBase Connectivity (JDBC) provides a generic solution for synchronizing data between Identity Manager and JDBC-accessible relational databases. The principal value of this driver resides in its generic nature. Unlike most drivers that interface with a single application, this driver can interface with most relational databases and database-hosted applications.

You can connect to a single database using a single JDBC driver. To configure a single JDBC driver to connect to multiple databases of the same type (for example, Oracle, MySQL, or PostgreSQL) use the driver with the Fan-Out agent. For more information, see OpenText™ Identity Manager Driver for JDBC Fanout Implementation Guide (https://www.netiq.com/documentation/identity-manager-4.10-drivers/jdbc_fanout/data/netiq-identity-manager-for-jdbc-fan-out-driver-implementation-guide.html).

- "Components for Data Synchronization" on page 13
- "Database Concepts" on page 16
- "How the Driver Works" on page 21
- "Supported Operations" on page 21
- "Planning to Install the Driver" on page 25

Components for Data Synchronization

This section provides information about the components required to integrate a connected system (database) with Identity Manager.

- "JDBC" on page 13
- "Third-Party JDBC Driver" on page 14
- "Identity Vault" on page 14
- "Schema" on page 14
- "Logical Database Class" on page 15
- "XDS" on page 15

JDBC

Java DataBase Connectivity (JDBC) is a cross-platform database interface standard that Sun Microsystems developed.

Most enterprise database vendors provide a unique implementation of the JDBC interface. Four versions of the JDBC interface are available:

- JDBC 1 (Java 1.0)

- JDBC 2 (Java 1.2 or 1.3)
- JDBC 3 (Java 1.4 or 1.5)
- JDBC 4 (Java 1.6 or 1.7)

The JDBC driver primarily uses the JDBC 1 interface. It uses a small subset of JDBC 2, JDBC 3, or JDBC 4 methods when supported by third-party JDBC drivers.

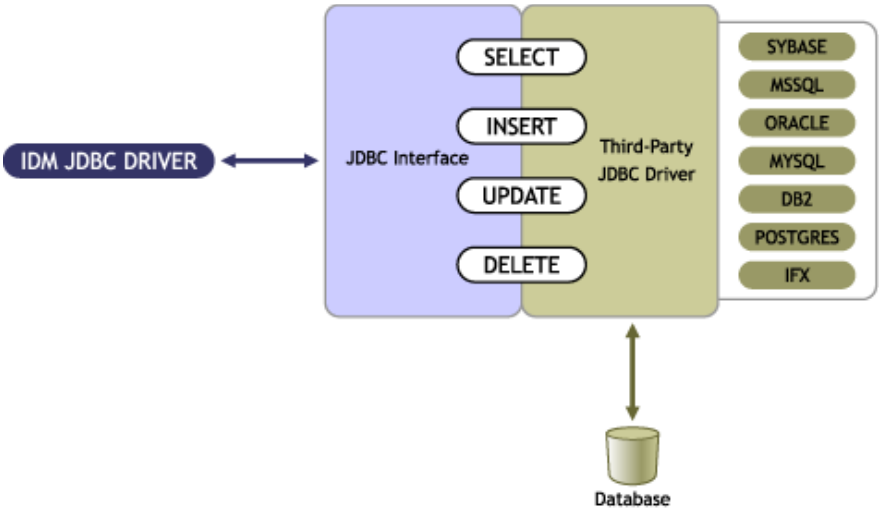
Third-Party JDBC Driver

A third-party JDBC driver is one of the numerous JDBC interface implementations that the Identity Manager JDBC driver uses to communicate with a particular database.

For example, `ojdbc6.jar` is one of the Oracle JDBC drivers. Different third-party JDBC drivers implement different portions of the JDBC interface specification and implement the interface in a relatively consistent manner.

The following illustration indicates the relationship between the Identity Manager JDBC driver and third-party JDBC drivers.

Figure 1-1 *Identity Manager JDBC Driver vs. Third-Party JDBC Drivers*



Identity Vault

An Identity Vault is the data store that Identity Manager uses.

The Identity Vault is a persistent database powered by OpenText eDirectory and used by Identity Manager to hold data for synchronization with a connected system. The vault can be viewed narrowly as a private data store for Identity Manager or more broadly as a metadirectory that holds enterprise-wide data. Data in the vault is available to any protocol supported by OpenText eDirectory, including NetWare Core Protocol (NCP) and LDAP.

Schema

- "Directory Schema" on page 15
- "Application Schema" on page 15

- "Database Schema" on page 15
- "Synchronization Schema" on page 15

Directory Schema

The directory schema is the set of object classes and attributes in the directory.

A database schema is a way to logically group objects such as tables, views, and stored procedures.

For example, the eDirectory User class and Given Name attribute are part of the eDirectory schema.

Application Schema

The application schema is the set of classes and attributes in an application.

Because databases have no concept of classes or attributes, the JDBC driver maps eDirectory classes to tables or views, and maps eDirectory attributes to columns.

Database Schema

Database schema is essentially synonymous with ownership. A database schema consists of database objects (for example, tables, views, triggers, stored procedures, and functions) that a database user owns.

With the JDBC driver, schema is useful to scope the database (reduce the number of database objects visible to the driver at runtime).

Ownership is often expressed by using a qualified dot notation (for example, `indirect.usr`, where `indirect` is the name of the database user that owns the table `usr`). All of the database objects owned by `indirect` constitute the indirect database schema.

Synchronization Schema

The synchronization schema is the database schema visible to the driver at runtime.

Logical Database Class

The logical database class is the set of tables or view used to represent an eDirectory class in a database.

XDS

XDS format is the defined NetIQ subset of possible XML formats that Identity Manager can use.

XDS is the initial format for data coming from the Identity Vault. By modifying default rules and changing the style sheets, you can configure the JDBC driver to work with any XML format.

Database Concepts

- "Structured Query Language" on page 16
- "Data Manipulation Language" on page 16
- "Data Definition Language" on page 16
- "View" on page 17
- "Identity Columns/Sequences" on page 17
- "Transaction" on page 18
- "Stored Procedures or Functions" on page 18
- "Trigger" on page 19
- "Instead-Of-Trigger" on page 20

Structured Query Language

Structured Query Language (SQL) is the language used to query and manipulate data in relational databases.

Data Manipulation Language

Data Manipulation Language (DML) statements are highly standardized SQL statements that manipulate database data.

DML statements are essentially the same, regardless of the database that you use. The JDBC driver is DML-based. It maps Identity Manager events expressed as XDS XML to standardized DML statements.

The following example shows several DML statements:

```
SELECT * FROM usr;
INSERT INTO usr(lname) VALUES('Doe');
UPDATE usr SET fname = 'John' WHERE idu = 1;
```

Data Definition Language

Data Definition Language (DDL) statements manipulate database objects such as tables, indexes, and user accounts.

DDL statements are proprietary and differ substantially between databases. Even though the JDBC driver is DML-based, you can embed DDL statements in XDS events. For additional information, refer to Chapter 12, "Embedded SQL Statements in XDS Events," on page 141,

The following examples show several DDL statements:

```
CREATE TABLE usr
(
    idu   INTEGER,
    fname VARCHAR2(64),
    lname VARCHAR2(64)
);

CREATE USER idm IDENTIFIED BY novell;
```

NOTE: Examples used throughout this guide are for the Oracle database.

View

A view is a logical table.

When queried by using a SELECT statement, the view is created by executing the SQL query supplied when the view was defined. Views are a useful abstraction mechanism for representing multiple tables of arbitrary structure as a single table or logical database class.

```
CREATE VIEW view_usr
(
    pk_idu,
    fname,
    lname
)
AS
SELECT idu, fname, lname from usr;
```

Identity Columns/Sequences

Identity columns and sequences are used to generate unique primary key values. Identity Manager can associate with these values, among other things.

An identity column is a self-incrementing column used to uniquely identify a row in a table. Identity column values are automatically filled in when a row is inserted into a table.

A sequence object is a counter that can be used to uniquely identify a row in a table. Unlike an identity column, a sequence object is not bound to a single table. However, if it is used by a single table, a sequence object can be used to achieve an equivalent result.

The following is an example of a sequence object:

```
CREATE SEQUENCE seq_idu
    START WITH 1
    INCREMENT BY 1
    NOMINVALUE
    NOMAXVALUE
    ORDER;
```

Transaction

A transaction is an atomic database operation that consists of one or more statements.

When a transaction is complete, all statements in the transaction are committed. When a transaction is interrupted or one of the statements in the transaction has an error, the transaction is said to roll back. When a transaction is rolled back, the database is left in the same state it was before the transaction began.

Transactions are either manual (user-defined) or automatic. Manual transactions can consist of one or more statements and must be explicitly committed. Automatic transactions consist of a single statement and are implicitly committed after each statement is executed.

- "Manual (User-Defined) Transactions" on page 18
- "Automatic Transactions" on page 18

Manual (User-Defined) Transactions

Manual transactions usually contain more than one statement. DDL statements typically cannot be grouped with DML statements in a manual transaction.

The following example illustrates a manual transaction:

```
SET AUTOCOMMIT OFF
INSERT INTO usr(lname) VALUES('Doe');
UPDATE usr SET fname = 'John' WHERE idu = 1;
COMMIT; -- explicit commit
```

Automatic Transactions

Automatic transactions consist of only one statement. They are often referred to as auto-committed statements because changes are implicitly committed after each statement. An auto-committed statement is autonomous of any other statement.

The following example illustrates an automatic transaction:

```
SET AUTOCOMMIT ON
INSERT INTO emp(lname) VALUES('Doe');
-- implicit commit
```

Stored Procedures or Functions

A stored procedure or function is programmatic logic stored in a database. Stored procedures or functions can be invoked from almost any context.

The Subscriber channel can use stored procedures or functions to retrieve primary key values from rows inserted into tables, to create associations. Stored procedures or functions can also be invoked from within embedded SQL statements or triggers.

The distinction between stored procedures and functions varies by database. Typically, both can return output, but they differ in how they do it. Stored procedures usually return values through parameters. Functions usually return values through a scalar return value or result set.

The following example illustrates a stored procedure definition that returns the next value of a sequence object:

```
CREATE SEQUENCE seq_idu
    START WITH 1
    INCREMENT BY 1
    NOMINVALUE
    NOMAXVALUE
    ORDER;

CREATE
PROCEDURE sp_idu(    io_idu IN OUT INTEGER)
IS
BEGIN
    IF (io_idu IS NULL) THEN
      SELECT seq_idu.nextval INTO io_idu FROM DUAL;
END IF;
END sp_idu;
```

Trigger

A database trigger is programmatic logic associated with a table, which executes under certain conditions. A trigger is said to fire when its execution criteria are met.

Triggers are often useful for creating side effects in a database. In the context of the JDBC driver, triggers are useful to capture event publications. The following is an example of a database trigger on the `usr` table.

```
CREATE TABLE usr
(
    idu   INTEGER,
    fname VARCHAR2(64),
    lname VARCHAR2(64)
);

-- t = trigger; i = insert
CREATE TRIGGER t_usr_i
    AFTER INSERT ON usr
    FOR EACH ROW

BEGIN
    UPDATE usr SET fname = 'John';
END;
```

When a statement is executed against a table with triggers, a trigger fires if the statement satisfies the conditions specified in the trigger. For example, using the above table, suppose the following insert statement is executed:

```
INSERT INTO usr(lname) VALUES('Doe')
```

Trigger `t_emp_i` fires after the insert statement is executed, and the following update statement is also executed:

```
UPDATE usr SET fname = 'John'
```

A trigger can typically be fired before or after the statement that triggered it. Statements that are executed as part of a database trigger are typically included in the same transaction as the triggering statement. In the above example, both the INSERT and UPDATE statements are committed or rolled back together.

Instead-Of-Trigger

An instead-of-trigger is programmatic logic associated with a view, which executes under certain conditions.

Instead-of-triggers are useful for making views writable or subscribeable. They are often used to define what it means to INSERT, UPDATE, and DELETE from a view. The following is an example of an instead-of-trigger on the usr table.

```
CREATE TABLE usr
(
    idu   INTEGER,
    fname VARCHAR2(64),
    lname VARCHAR2(64)
);

CREATE VIEW view_usr
(
    pk_idu,
    fname,
    lname
)
AS
SELECT idu, fname, lname from usr;
-- t = trigger; i = insert
CREATE TRIGGER t_view_usr_i
    INSTEAD OF INSERT ON usr
BEGIN
    INSERT INTO usr(idu, fname, lname)
      VALUES(:NEW.pk_idu, :NEW.fname, :NEW.lname);
END;
```

When a statement is executed against a view with instead-of-triggers, an instead-of-trigger executes if the statement satisfies the conditions specified in the trigger. Unlike triggers, instead-of-triggers always execute before the triggering statement. Also, unlike regular triggers, instead-of-triggers are executed instead of, not in addition to, the triggering statement.

For example, using the above view, suppose the following insert statement is executed instead of the original insert statement:

```
INSERT INTO view_usr(pk_idu, fname, lname)
    VALUES(1, ‘John', ‘Doe')
```

Rather than executing the original statement, instead-of-trigger t_view_usr_i fires and executes the following statement:

```
INSERT INTO usr(idu, fname, lname)
    VALUES(:NEW.pk_idu, :NEW.fname, :NEW.lname);
```

In this example, the statements happen to be equivalent.

How the Driver Works

- "Generic JDBC Driver" on page 21

Generic JDBC Driver

The Identity Manager JDBC driver uses the JDBC interface to synchronize data and identities between an Identity Vault and relational databases.

The driver consists of the following jar files:

- `JDBCShim.jar`
- `JDBCUtil.jar`
- `JDBCConfig.jar`

In addition to these files, you need a third-party JDBC driver to communicate with every database.

Supported Operations

- "Supported Databases" on page 21
- "Supported Third Party JDBC Drivers" on page 21
- "Support for Password Synchronization" on page 21
- "Supported Data Synchronization Models" on page 21
- "Triggerless vs. Triggered Publication" on page 24

Supported Databases

For information on supported databases, see"Supported Databases" on page 167.

Supported Third Party JDBC Drivers

For information on supported third-party JDBC drivers, see "Third-Party JDBC Driver Interoperability" on page 181.

Support for Password Synchronization

The JDBC driver supports password set and check on the Subscriber channel. The driver does not support bidirectional password synchronization.

Supported Data Synchronization Models

The JDBC driver supports two data synchronization models: direct and indirect. Both terms are best understood with respect to the final destination of the data being synchronized.

Model	Association	Description
Indirect	Usually associated with tables	Target database tables probably don't match the structure required by the driver. Therefore, it's usually necessary to create intermediate staging tables that do match the structure that the driver requires.
Direct	Usually associated with views	Views provide the abstraction mechanism that best facilitates integration with the target database tables.

The following sections describe how direct and indirect synchronization work on both the Subscriber and Publisher channels.

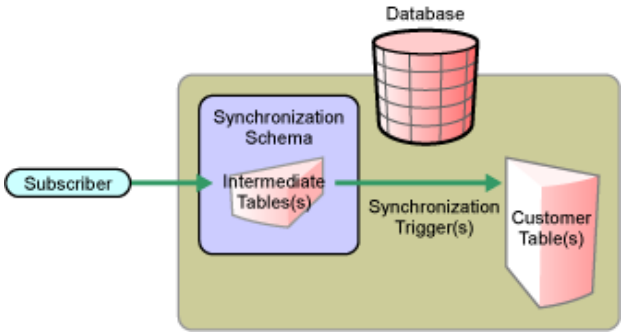
- "Indirect Synchronization" on page 22
- "Direct Synchronization" on page 23

Indirect Synchronization

Indirect synchronization uses intermediate staging tables to synchronize data between the Identity Vault and a database. You can have one or more customer tables and intermediate staging tables.

Subscriber Channel

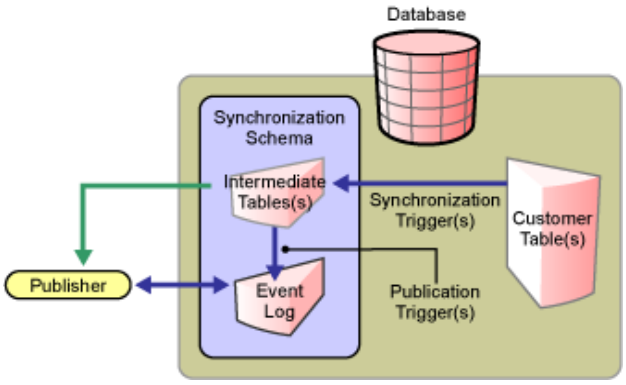
The Subscriber channel updates the intermediate staging tables in the synchronization schema. This action triggers an update to customer tables elsewhere in the database.



Publisher Channel

Synchronization triggers update the intermediate staging tables when target database tables are updated. Publication triggers then insert one or more rows into the event log table. The Publisher channel then reads the inserted rows and updates the Identity Vault.

Depending on the contents of the rows read from the event log table, the Publisher channel might need to retrieve additional information from the intermediate tables before updating the Identity Vault. After updating the Identity Vault, the Publisher channel then deletes or marks the rows as processed.



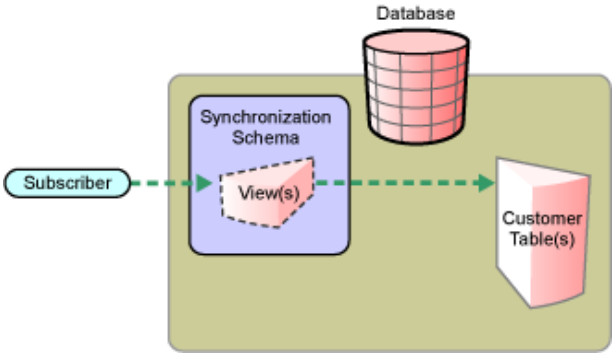
Direct Synchronization

Direct synchronization typically uses views to synchronize data between Identity Manager and a database. You can use tables if they conform to the structure that the JDBC driver requires. You can have one or more customer views or tables.

Subscriber Channel

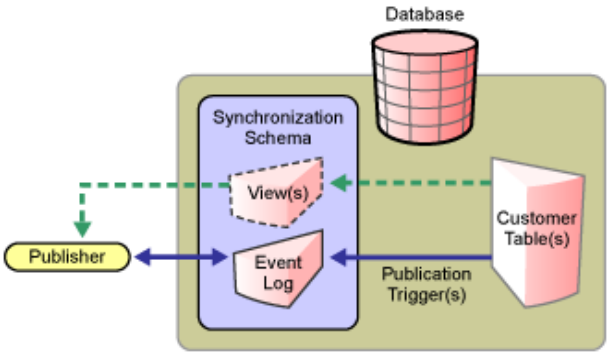
Updates existing customer tables through a view in the synchronization schema.

Direct synchronization without a view is possible only if the target database tables match the structure that the JDBC driver requires. For additional information, see "Indirect Synchronization" on page 111.



Publisher Channel

When a target database table is updated, publication triggers insert rows into the event log table. The Publisher channel then reads the inserted rows and updates the Identity Vault.



Depending on the contents of the rows read from the event log table, the Publisher channel might need to retrieve additional information from the view before updating the Identity Vault. After updating the Identity Vault, the Publisher channel then deletes or marks the rows as processed.

Triggerless vs. Triggered Publication

Triggers are not required to log events for the Publisher channel. In situations where triggers cannot be used to capture granular events, the Publisher channel can derive database changes by inspecting database data.

Triggerless publication is particularly useful when support contracts forbid the use of triggers on database application tables or for rapid prototyping.

However, triggerless publication is less efficient than triggered publication. With triggered publication, what changed is already known. With triggerless publication, change calculation must occur before events can be processed.

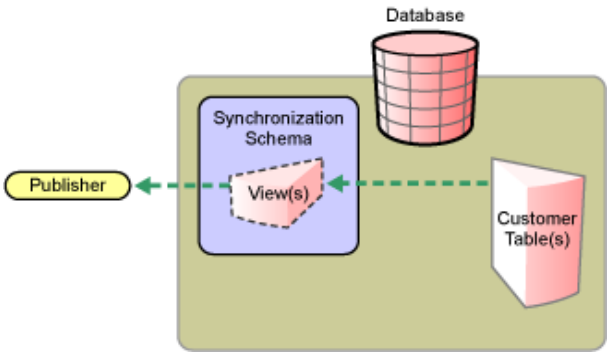
Triggerless publication, unlike triggered publication, does not preserve event order. It only guarantees that, by the end of a polling cycle, objects in the database and the Identity Vault are in sync.

Triggerless publication, unlike triggered publication, does not provide historical data such as old values. It provides information on the current state of an object, not the previous state.

Triggerless publication does have the advantage of being much simpler because it reduces database-side dependencies. Writing database triggers can be complicated and requires extensive knowledge of database-specific SQL syntaxes.

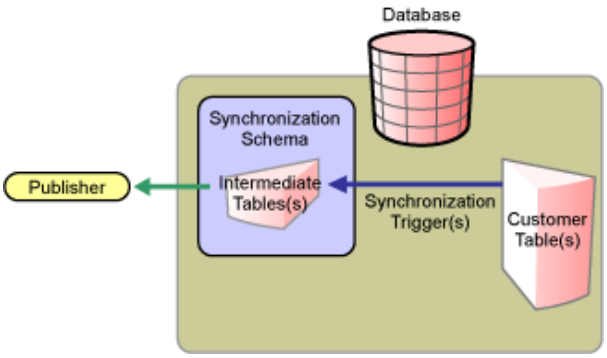
The following figure illustrates direct triggerless publication:

Figure 1-2 Direct Triggerless Publication



The following figure illustrates indirect triggerless publication:

Figure 1-3 Indirect Triggerless Publication



If you move the driver without moving the state files, the driver must build up new state files by resynchronizing. For information on this situation, see "State Directory" on page 62.

Planning to Install the Driver

This section provides information for planning the installation and configuration process for the driver.

Installation Requirements

For information about installing the driver, see "Installing the Driver Files" on page 27.

Options for Installing the Driver Shim

You can install the driver shim on the Identity Manager server. Alternatively, you can use the Remote Loader service to run the driver on a server other than the Identity Manager server. In this case, the driver and the Remote Loader service run on the same server. The Remote Loader loads drivers and communicates with the Identity Manager engine on behalf of drivers installed on remote servers. For information about the operating systems supported for Identity Manager engine and Remote Loader, see the OpenText Identity Manager Technical Information website (https://www.netiq.com/products/identity-manager/advanced/technical-information/).

2 Installing the Driver Files

By default, the JDBC driver files are installed on the Identity Manager server at the same time as the Identity Manager engine. The installation program extends the Identity Vault's schema and installs the driver shim and a driver configuration file. It does not create the driver object in the Identity Vault (see Chapter 5, "Creating a New Driver Object," on page 43) or upgrade an existing driver's configuration (see Chapter 7, "Upgrading an Existing Driver," on page 103).

The JDBC driver can either be located on the same server as the JDBC database or any other server. The following sections explain what to do if the JDBC driver files are not on the JDBC database server and how to install the third-party JDBC jar files that the driver uses to communicate with the database:

For information about uninstalling the driver, see Appendix A, "Uninstalling the Driver," on page 209.

- "Installing the Driver Files" on page 27
- "Installing JDBC Driver Jar Files" on page 27

Installing the Driver Files

You can install the JDBC driver files in the following ways:

- On a local machine: Install the JDBC driver files on the Identity Manager server and connect to the database by using the Provider URL (Connection Properties).
- On a remote machine: Install the JDBC driver files on the Remote Loader.

For information on installing the Identity Manager server or the Remote Loader, see Considerations for Installing Identity Manager Engine Components and Remote Loader in the *OpenText™ Identity Manager Install and Upgrade Guide for Linux (https://www.netiq.com/documentation/identity-manager-4.10/setup_linux/data/front.html)* or Planning to Install the Engine, Drivers, and Plug-ins in the.*OpenText™ Identity Manager CE 24.4 (v4.10) Install and Upgrade Guide for Windows (https://www.netiq.com/documentation/identity-manager-4.10/setup_windows/data/front.html)*.

Installing JDBC Driver Jar Files

To communicate with the JDBC database, the JDBC driver requires that you copy the appropriate JDBC driver jar files to the driver location.

1. Locate the appropriate JDBC driver jar files.

   Information about the jar files you need and where to download them from is found in "Supported Third-Party JDBC Drivers (Recommended)" on page 182.

2. Place the files in the appropriate location.

   The following tables identify the paths where you need to place JDBC driver jar files on a Identity Manager server or on a Remote Loader server that is running the JDBC driver.

Table 2-1 Locations for JAR Files: Identity Manager Server

Platform	Directory Path
Solaris, Linux, or AIX	`/opt/novell/eDirectory/lib/dirxml/classes`
Windows	`novell\NDS\lib`

Table 2-2 Locations for JAR Files: Remote Loader

Platform	Directory Path
Solaris, Linux, or AIX	`/opt/novell/eDirectory/lib/dirxml/classes`
Windows	`novell\RemoteLoader\lib`

3 The Association Utility

The Association utility normalizes associations of objects associated under the 1.0 or later versions of the JDBC driver. It also provides several other features that simplify driver administration.

This version of the utility is compatible with the 1.0 and later versions of the JDBC driver, and supersedes all previous versions.

- "Independent Operations" on page 29
- "Before You Begin" on page 30
- "Using the Association Utility" on page 31
- "Parameters for Searching and Replacing" on page 32

Independent Operations

The Association utility supports seven independent operations:

Table 3-1 *Independent Operations*

Operation	Description	Read-Write Functionality
1	Lists objects associated with a driver (default).	Read-only
2	Lists objects that have multiple associations to a driver.	Read-only
3	Lists objects that have invalid associations to a driver. An association is invalid if: ◆ It is malformed. For example, the association is missing the schema RDN, missing the table RDN, or the schema keyword is misspelled. ◆ It contains database identifiers that do not map to identifiers in the target database. For example, an association includes a mapping to a table that does not exist. ◆ It maps to no row or multiple rows. An association is broken if it doesn't map to a row. Also, associations aren't unique if they map to more than one row.	Read-only
4	Lists objects that need to be normalized. A normalized association is valid, correctly ordered, and uses the correct case. Normal case is uppercase for case-insensitive databases and mixed case for case-sensitive databases.	Read-only

Operation	Description	Read-Write Functionality
5	Normalizes object associations listed during operation 4.	Write
6	Lists object associations to be modified. Allows for global replacement of schema, table, and column names based on search criteria. This operation requires two parameters (oldRDN and newRDN). See "Parameters for Searching and Replacing" on page 32.	Read-only
7	Modifies object associations listed during operation 6. This operation requires two parameters (oldRDN and newRDN). See "Parameters for Searching and Replacing" on page 32.	Write

Before You Begin

Modifying associations can cause problems. If associations are corrupted, Identity Manager ceases to function. Therefore, use write operations only when necessary. To avoid unintentionally corrupting an association, the Association utility creates an undo ldiff file for all write operations.

Review the following cautions before using the utility:

- The Association utility, like the driver, assumes that database identifiers are undelimited (unquoted and contain no special characters).
- Update all object associations related to a driver at the same time.

  Updating associations at the same time is extremely important.

  To see all of the objects associated with a particular driver, run the Association utility on the Identity Manager server associated with a particular driver instance.

  The LDAP search base must contain all of the objects associated with a particular driver.

  To ensure complete containment, we recommend that you use your tree's root container as the search base.
- Make sure that the JDBC URL of the target database supplied to this utility is the same as the URL that the driver uses. Pointing this utility at a case-insensitive database when the database is actually case-sensitive might result in associations being normalized to the wrong case.
- Because the Association utility runs locally, it uses an unsecured connection. Therefore, the Identity Vault LDAP server must be temporarily configured to accept clear text passwords. Depending upon the third-party JDBC driver you are using, the database connection established by this utility might not be secure.

  We recommend changing the driver's authentication password on the database after you run this utility.

Using the Association Utility

Run the Association utility once for each instance of the driver installed on an Identity Manager server. In the *install-dir*\DirXMLUtilities\jdbc\util directory, a batch file association.bat or shell script association.sh (depending upon your platform) starts the utility.

A properties file containing association utility parameters is provided for each supported database. These files are in the *install-dir*\DirXMLUtilities\jdbc\util directory.

Table 3-2 *Properties Files*

Database	Properties Filename
IBM DB2 Universal Database	properties_db2.txt
Informix Dynamic Server	properties_ifx_ansi.txt1 properties_ifx_log.txt properties_ifx_no_log.txt
Microsoft SQL Server	properties_ms.txt
MySQL	properties_my.txt
Oracle	properties_ora.txt
PostgreSQL	properties_pg.txt
Sybase Adaptive Server Enterprise	properties_syb.txt

This utility does not work with Informix ANSI-compliant databases.

1. Stop the driver.
2. Use association.bat or association.sh to run the Association utility to identify and remove extraneous associations (operations 2 and 3).

   No object associated by this product should have multiple associations. Manually remove extraneous associations on a per object basis. Operation 3 might help you identify which of the multiple associations is actually valid. After you know this, you can probably discard the extraneous associations.
3. Run the Association utility to identify and fix invalid associations (operation 3 and possibly operations 6 and 7).

   As a general rule, if the problem is isolated, manually edit each invalid association. If the problem is repetitive and affects a large number of associations, consider using operations 6 and 7. This utility can replace bad identifiers on a global basis, but cannot insert or remove them where they do not already exist. See "Parameters for Searching and Replacing" on page 32 for information about search parameters.
4. Run the Association utility to normalize associations (operations 4 and 5).

Parameters for Searching and Replacing

The Association utility requires two parameters (oldRDN and newRDN) for operations 6 and 7 in order to search and replace.

The first value (for example, schema) in the parameter is the search criterion. The second value (for example, old) is the replacement value. Under certain scenarios, you can use the wildcard character `*` to generalize the search criterion or replacement value.

Three types of search and replace operations are possible:

Table 3-3 *Search and Replace Operations*

Option	Description	Example
Replace the schema name	Replace schema `old` with schema `new`. Wildcards are supported on the right side only.	oldRDN: `schema=old` newRDN: `schema=new`
Replace the table name	Replace table `old` with table `new`. Wildcards are not supported.	oldRDN: `table=old` newRDN: `table=new`
Replace the column name	Replace column `old` with column `new`. Wildcards are required on the right side, but they aren't supported on the left side.	oldRDN: `old=*` newRDN: `new=*`

4 Installing and Configuring Database Objects

You need to install and configure database objects (for example, tables, triggers, and indexes) for synchronization with the sample driver configuration. If you do not configure the database objects, the sample configuration file will not work.

- "SQL Script Conventions" on page 33
- "Installing IBM DB2 Universal Database (UDB)" on page 35
- "Installing Informix Dynamic Server (IDS)" on page 36
- "Installing Microsoft SQL Server" on page 36
- "Installing MySQL" on page 37
- "Installing Oracle" on page 37
- "Installing PostgreSQL 8" on page 37
- "Installing PostgreSQL 9" on page 38
- "Installing Azure Database for PostgreSQL 14" on page 39
- "Installing Sybase Adaptive Server Enterprise (ASE)" on page 40
- "Testing the Database Object Installation" on page 40

SQL Script Conventions

The following table lists default locations for SQL scripts:

Table 4-1 *Default Locations for SQL Scripts*

Platform	Default Location
Windows	`c:\Novell\IdentityManager\NDS\DirXMLUtilities\jdbc\sql\`
UNIX/Linux	`/opt/novell/eDirectory/lib/dirxml/rules/jdbc/sql/`

For example, when the scripts are installed on a SUSE Linux Enterprise Server with eDirectory, the DB2 scripts are found in `opt/novell/eDirectory/lib/dirxml/rules/jdbc/db2_udb/install` directory.

All SQL scripts use the same conventions, regardless of the database.

The maximum size of a DB2 identifier is 18 characters. This least common denominator length defines the upper bound of database identifier length across all SQL scripts. Because of this restricted length, abbreviations are used. The following table summarizes identifier abbreviations and their meanings:

Table 4-2 Identifier Abbreviations and Meanings

Abbreviation	Interpretation
proc_	stored procedure/function
idx_	index
trg_	trigger
_i	on insert trigger
_u	on update trigger
_d	on delete trigger
chk_	check constraint
pk_	view primary key constraint
fk_	view foreign key constraint
mv_	view multi-valued column
sv_	view single-valued column (implicit default)

Instead of `proc_`, the more common abbreviation is `sp_`. This prefix is reserved for system-stored procedures on Microsoft SQL Server. Also, this prefix forces lookup of a procedure first in the master database before evaluating any qualifiers (for example, database or owner). To maximize procedure lookup efficiency, this prefix has been deliberately avoided.

The following table indicates identifier naming conventions for indexes, triggers, stored procedures, functions, and constraints:

Table 4-3 Identifier Naming Conventions

Database Object	Naming Convention	Examples
stored procedure/ function	proc_*procedure-or-function-name*	proc_idu
index	idx_*unqualified-table-name_sequence-number*	idx_indirectlog_ 1
trigger	tgr_*unqualified-table-name_triggering-statement-type_sequence-number*	tgr_usr_i_1
primary key constraint	pk_*unqualified-table-name_column-name*	pk_usr_idu
foreign key constraint	fk_*unqualified-table-name_column-name*	fk_usr_idu
check constraint	chk_*unqualified-table-name_column-name*	chk_usr_idu

Other conventions:

- All database identifiers are lowercase.

  This is the most commonly used case convention between databases.

- String field lengths are 64 characters.

  Fields of this length can hold most eDirectory attribute values. You might want to refine field lengths to enhance storage efficiency.
- For performance reasons, primary key columns use native, scalar numeric types whenever possible (such as `BIGINT` as opposed to `NUMERIC`).
- The `record_id` column in event log tables has the maximum numeric range permitted by each database to avoid overflow.
- Identity columns and sequence objects do not cache values. Some databases throw away cached values when a rollback occurs. This action can cause large gaps in identity column or sequence values.

Installing IBM DB2 Universal Database (UDB)

IMPORTANT: For IBM DB2, you must manually create operating system user accounts before running the provided SQL scripts.

Because the process to create user accounts differs between operating systems, Step 1 below is OS-specific. These instructions are for a Windows NT operating environment. If you rerun the SQL scripts, repeat only Step 2 through Step 4.

The directory context for DB2 is found in the *`install-dir`*`\DirXMLUtilities\jdbc\sql\db2_udb\install` directory on Windows or *`install-dir`*`/lib/dirxml/rules/jdbc/sql/db2_udb/install` directory on UNIX/Linux platforms.

1. Create user accounts for users `idm`, `indirect` and `direct`.

   Use `novell` as the password in **User Manager for Domains**.

   Remember to deselect **User Must Change Password at Next Login** for this account.

   You might want to also select **Password Never Expires**.

   NOTE: The remaining instructions are OS-independent.

   ---

2. Adjust the file path to `idm_db2.jar` in the `1_install.sql` installation script. The file path to `idm_db2.jar` should reflect the location of this file on your client machine.
3. Execute the `1_install.sql` script from the Command Line Processor (CLP.)

   For example: `db2 -f 1_install.sql`

   IMPORTANT: The scripts won't execute in the Command Center interface beyond version 7. The scripts use \ as the line continuation character. Later versions of the Command Center don't recognize this character.

   ---

4. For versions 9 or later, execute the `2_install_9.sql` script.

   For example: `db2 -f 2_install_9.sql`

Installing Informix Dynamic Server (IDS)

For Informix Dynamic Server, you must manually create an operating system user account before running the provided SQL scripts.

Because the process of creating user accounts differs between operating systems, Step 1 below is OS-specific. These instructions are for a Windows operating environment. If you rerun the SQL scripts, you should repeat only Steps 2 through 4.

The directory context for Informix SQL scripts is found in the *install-dir*\DirXMLUtilities\jdbc\sql\informix_ids\install on Windows or *install-dir*/lib/dirxml/rules/jdbc/sql/informix_ids/install directory on UNIX/Linux platforms.

1. In Windows, create a user account for user idm.

   Use novell as the password in **User Manager for Domains**.

   Remember to deselect **User Must Change Password at Next Login** for this account.

   You might want to also select **Password Never Expires**.

   NOTE: The remaining instructions are OS-independent.

   ---

2. Start a client such as SQL Editor or DBAccess.
3. Log in to your server as the informix user or another user with DBA (database administrator) privileges.

   By default, the password for the informix user is informix. If you execute scripts as a user other than informix, change all references to informix in the scripts prior to execution.
4. Open and execute 1_install_9.sql from either the ansi (transactional, ANSI-compliant), log (transactional, non-ANSI-compliant), or no_log (non-transactional, non-ANSI-compliant) subdirectory, depending upon which type of database you want to create.
5. For version 11 or later, open and execute 2_install_10.sql from either the ansi (transactional, ANSI-compliant), log (transactional, non-ANSI-compliant), or no_log (non-transactional, non-ANSI-compliant) subdirectory, depending upon which type of database you want to create.

Installing Microsoft SQL Server

The directory context for Microsoft SQL Server scripts is found in the *install-dir*\DirXMLUtilities\jdbc\sql\mssql\install directory on Windows or *install-dir*/lib/dirxml/rules/jdbc/sql/mssql/install directory on UNIX/Linux platforms.

1. Start a client. For example, Microsoft SQL Server Management Studio.
2. Log in to your database server as the sa user.

   By default, the sa user has no password.
3. Execute the installation script.

   For version 2005, execute 1_install_2005.sql.

   For version 2008, execute 1_install_2005.sql.

   For version 2008 R2, execute 1_install_2005.sql.

Installing MySQL

The directory context for MySQL SQL scripts is found in the `\products\IDM\windows\setup\drivers\jdbc\tools\sql\` directory on Windows. On Linux platforms, the files are deployed at `/opt/novell/eDirectory/lib/dirxml/rules/jdbc/sql` when `novell-DXMLjdbc-4.1.0-0.noarch.rpm` is installed.

1. From a MySQL client, such as mysql, log in as root user or another user with administrative privileges.

   For example, from the command line, execute

   ```
   mysql -u root -p
   ```

   By default, the `root` user has no password.

2. Execute the installation script `1_install_innodb.sql` or `1_install_myisam.sql`, depending upon which table type you wish to use. For version 5.5.15 use the scripts in subdirectory 5.

   For example: `mysql> \. c:\1_install_innodb.sql`

   Don't use a semicolon to terminate this statement.

Installing Oracle

The directory context for Oracle SQL scripts is found in the *`install-dir`*`\DirXMLUtilities\jdbc\sql\oracle\install` directory on Windows or *`install-dir/`*`lib/dirxml/rules/jdbc/sql/oracle/install` directory on UNIX/ Linux platforms.

1. From an Oracle client, such as SQL Plus, log in as the SYSTEM user.

   By default, the password for SYSTEM is MANAGER. If you execute scripts as a user other than SYSTEM with password MANAGER, change all references to SYSTEM in the scripts prior to execution.

2. Execute the installation script `1_install.sql`.

   For example: `SQL> @c:\1_install.sql`

Installing PostgreSQL 8

The directory context for PostgreSQL scripts is found in the *`install-dir`*`\DirXMLUtilities\jdbc\sql\postgres\install` directory on Windows or *`install-dir`*`/lib/dirxml/rules/jdbc/sql/postgres/install` directory on UNIX/Linux platforms. The directory context for executing PostgreSQL commands is *`postgres-install-dir/`*`pgsql/bin`.

1. Create the `idm` database.

   For example, from the UNIX command line, execute the following command:

   ```
   ./createdb idm
   ```

2. From a PostgreSQL client such as psql, log in to the `idm` database as `postgres` user.

   For example, from the UNIX command line, execute the following command:

   ```
   ./psql -d idm postgres
   ```

By default, the `postgres` user has no password.

3 From inside psql, execute the script `1_install_8.sql`. For example:

```
idm=# \i 1_install_8.sql
```

4 Update the `pg_hba.conf` file.

In PostgreSQL 8, you can update the file through `pgAdminIII`. After you start, go to **Tools** > **Connect** to connect to the server, select the `idm` database, then go to **Tools** > **Server Configuration** > `pg_hba.conf`. In the pgAdminIII pg_hba.conf editor, the **IP-ADDRESS** and **IP-MASK** columns in the file are combined into a single field named as **IP-Address**. Place the values of **IP-ADDRESS** and **IP-MASK** columns in this field separated by a single whitespace character.

For example, add entries for the `idm` database user. Adjust the IP-ADDRESS and IP-MASK as necessary:

```
# TYPE DATABASE    USER   IP-ADDRESS        IP-MASK           METHOD#
allow driver user idm to connect to database idm
host    idm     idm    <ip-address>   <net-mask>     password
```

5 Restart the PostgreSQL server to effect changes made to the `pg_hba.conf` file.

6 (Conditional) If you are using pgAdminIII, in the pg_hba.conf editor select the disk icon (save file) in the toolbar. When prompted, press **Yes**.

Installing PostgreSQL 9

The directory context for PostgreSQL scripts is found in the *install-dir*`\DirXMLUtilities\jdbc\sql\postgres\install` directory on Windows or *install-dir*`/lib/dirxml/rules/jdbc/sql/postgres/install` directory on UNIX/Linux platforms. The directory context for executing PostgreSQL commands is *postgres-install-dir*`/pgsql/bin`.

1 Create the `idm` database.

For example, from the UNIX command line, execute the following command:

```
./createdb idm
```

2 Install the `plpgsql` procedural language to the `idm` database.

For example, from the UNIX command line, execute the following command:

```
./createlang plpgsql idm
```

3 From a PostgreSQL client such as psql, log in to the `idm` database as `postgres` user.

For example, from the UNIX command line, execute the following command:

```
./psql -d idm postgres
```

By default, the `postgres` user has no password.

4 From inside psql, execute the script `1_install_8.sql`. For example:

```
idm=# \i 1_install_8.sql
```

5 Update the `pg_hba.conf` file.

For example, add entries for the `idm` database user. Adjust the **IP-ADDRESS** and **IP-MASK** columns as necessary:

```
# TYPE  DATABASE    USER   IP-ADDRESS        IP-MASK           METHOD#
allow driver user idm to connect to database idm
host    idm         idm    <ip-address>   <net-mask>     password
```

6 Restart the PostgreSQL server to effect the changes made to the `pg_hba.conf` file.

Installing Azure Database for PostgreSQL 14

The directory context for PostgreSQL scripts is found in the *`install-dir`*`\DirXMLUtilities\jdbc\sql\postgres\install` directory on Windows or *`install-dir`*`/lib/dirxml/rules/jdbc/sql/postgres/install` directory on UNIX/Linux platforms. The directory context for executing PostgreSQL commands is *`postgres-install-dir`*`/pgsql/bin`.

1 Login to Azure Portal.

1a Goto **Network Settings** of PostgreSQL server.

1b Download the SSL certificates.

1c Goto **Firewall Settings**, click **Add** and provide the IP address as - 0.0.0.0.

1d Open PGADMIN and add Server

1e Navigate to **General Tab**.

1f Navigate to **Connections Tab**, provide **hostname** as shown in the format below:

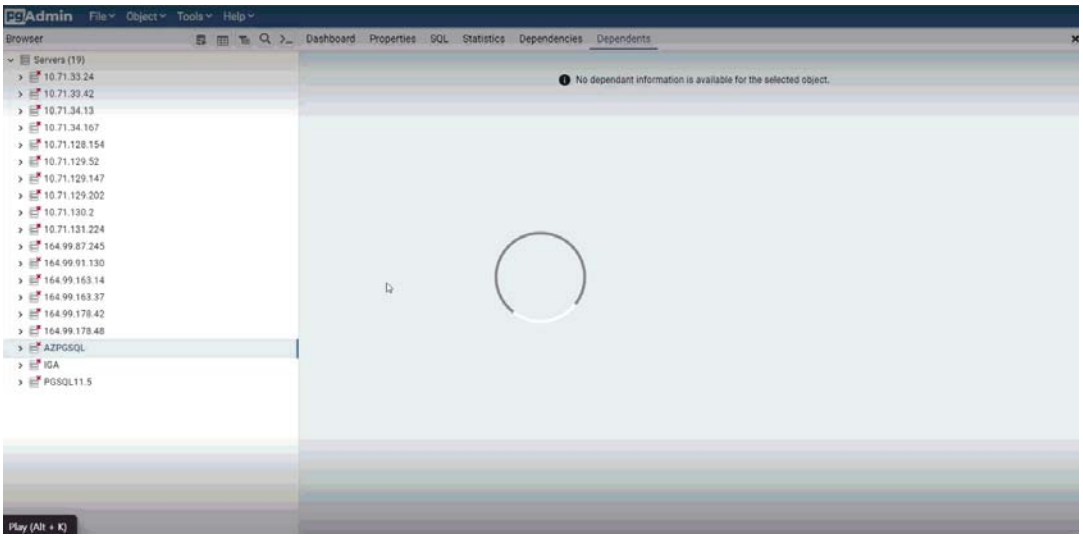
<pgsqlsrvr>.postgres.database.azure.com

For example: pgsqlsrvr.postgres.database.azure.com

1g Specify the **Port** number. By default the port number is 5432 and specify the **Maintenance Database** value as Postgres.

1h Click on **SSL** tab. In **Root Certificate** field, provide the path for the certificate that has been downloaded in Step 1c on page 39 from Azure Portal as shown in the below image.

Figure 4-1



2 Running the `install` script.

   2a Connect to Postgres database using PGADMIN tool.

   2b Right click on the Postgres database and click on Query tool.

   2c In the Query tool run the commands in the `install` scripts for creating a user and granting permissions / privileges to the user created.

   2d Logout of PGADMIN and login as the user created in Step 2c on page 40. Right Click on the Posgres database and click on Query tool. User-name should be the Database Administrator which gets created as part of deployment of Azure Database for PostgreSQL flexible server.

   2e From the `install` script copy all the contents from CREATE A DATABASE till the end of the script. Paste the copied content in the Query Tool and run the query.

Installing Sybase Adaptive Server Enterprise (ASE)

The directory context for Sybase SQL scripts is found in the *`install-dir`*`\DirXMLUtilities\jdbc\sql\sybase_ase\install` directory on Windows or *`install-dir`*`/lib/dirxml/rules/jdbc/sql/sybase_ase/install` directory on UNIX/Linux platforms.

1 From a Sybase client, such as isql, log in as the `sa` user and execute the `1_install.sql` installation script.

   For example, from the command line, execute:

```
isql -U sa -P -i 1_install.sql
```

   By default, the sa account has no password.

Testing the Database Object Installation

Test scripts for each database are located in the following directories:

Table 4-4 *Location of Database Scripts*

Database	Test SQL Scripts Location
IBM DB2 Universal Database	**UNIX/Linux:** *`install-dir`*`/lib/dirxml/rules/jdbc/sql/db2_udb/test` **Windows:** *`install-dir`*`\DirXMLUtilities\jdbc\sql\db2_udb\test`

Database	Test SQL Scripts Location
Informix Dynamic Server	**UNIX/Linux:** The test scripts are located in the following directories: ◆ *install-dir*/lib/dirxml/rules/jdbc/sql/informix_ids/test/log ◆ *install-dir*/lib/dirxml/rules/jdbc/sql/informix_ids/test/no_log **Windows:** The test scripts are located in the following directories: ◆ *install-dir*\DirXMLUtilities\jdbc\sql\informix_ids\log\test ◆ *install-dir*\DirXMLUtilities\jdbc\sql\informix_ids\no_log\test **Informix:** Informix ANSI test scripts are located in the log subdirectory.
Microsoft SQL Server	**UNIX/Linux:** The test scripts are located in the following directories: ◆ *install-dir*/lib/dirxml/rules/jdbc/sql/mssql/3or4/test ◆ *install-dir*/lib/dirxml/rules/jdbc/sql/mssql/5/test **Windows:** *install-dir*\DirXMLUtilities\jdbc\sql\mssql\test
MySQL	**UNIX/Linux:** The test scripts are located in the following directories: ◆ *install-dir*/lib/dirxml/rules/jdbc/sql/mysql/3or4/test ◆ *install-dir*/lib/dirxml/rules/jdbc/sql/mysql/5/test **Windows:** *install-dir*\DirXMLUtilities\jdbc\sql\mysql\test
Oracle	**UNIX/Linux:** *install-dir*/lib/dirxml/rules/jdbc/sql/oracle/test **Windows:** *install-dir*\DirXMLUtilities\jdbc\sql\oracle\test
PostgreSQL	**UNIX/Linux:** *install-dir*/lib/dirxml/rules/jdbc/sql/postgres/test **Windows:** *install-dir*\DirXMLUtilities\jdbc\sql\postgres\test

Database	Test SQL Scripts Location
Sybase Adaptive Server Enterprise	**UNIX/Linux:** *install-dir*/lib/dirxml/rules/jdbc/sql/sybase_ase/test **Windows:** *install-dir*\DirXMLUtilities\jdbc\sql\sybase_ase\test

You should try the test scripts before starting the sample driver.

If you encounter issues while testing, see the following sections:

- "Recognizing Publication Events" on page 206.
- "Executing Test Scripts" on page 207.

5 Creating a New Driver Object

After the JDBC driver files are installed on the server where you want to run the driver object (see Chapter 2, "Installing the Driver Files," on page 27), you can create a driver object in the Identity Vault. You do so by installing the driver packages and then modifying the driver configuration to suit your environment. The following sections provide instructions:

- "Creating the Driver Object in Designer" on page 43
- "Activating the Driver" on page 49
- "JDBC Driver Settings" on page 50

Creating the Driver Object in Designer

You create the JDBC driver by installing the driver packages and then modifying the configuration to suit your environment. After you create and configure the driver object, you need to deploy it to the Identity Vault and start it.

- "Importing the Current Driver Packages" on page 43
- "Installing the Driver Packages" on page 44
- "Configuring the Driver Object" on page 48
- "Deploying the Driver Object" on page 48
- "Starting the Driver Object" on page 49

NOTE: To create drivers, you need to use the new package management features provided in Designer.

Importing the Current Driver Packages

The driver packages contain the items required to create a driver object, such as policies, entitlements, filters, and Schema Mapping policies. These packages are only available in Designer. You can upgrade any package that is installed if there is a newer version of the package available. For more information on upgrading packages, see "Upgrading Installed Packages (https://www.netiq.com/documentation/identity-manager-4.10/designer_admin/data/packman.html#packmanupgrade)" in the *OpenText™ Designer for Identity Manager Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/designer_admin/data/bookinfo.html)*.

Before creating a driver object in Designer, you need to verify that you have all the required packages already imported in the Package Catalog of Designer. Designer prompts you for importing the required packages when it creates the driver object.

You can create packages based on the schema for your environment, keeping in mind the data synchronization model (direct/indirect) and its dependent packages.

To verify you have the most recent version of the driver packages in the Package Catalog:

1 Open Designer.

2 In the toolbar, click **Help** > **Check for Package Updates**.

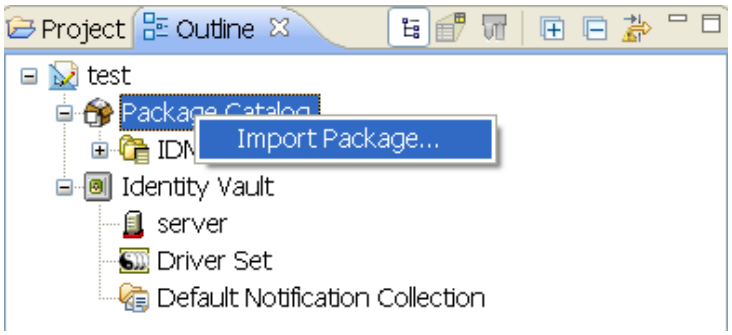
3 Click **OK** to update the packages

or

Click **OK** if the packages are up-to-date.

4 In the Outline view, right-click the Package Catalog.

5 Click **Import Package**.



You can download the new packages from the *Designer Auto-update site* (http://cdn.novell.com/cached/designer/packages/idm/updatesite1_0_0/).

6 Select any JDBC driver packages.

or

Click **Select All** to import all of the packages displayed.

By default, only the base packages are displayed. Deselect **Show Base Packages Only** to display all packages.

7 Click **OK** to import the selected packages, then click **OK** in the successfully imported packages message.

8 After the current packages are imported, continue with "Installing the Driver Packages" on page 44.

Installing the Driver Packages

After you have imported the current driver packages into the Package Catalog, you can install the driver packages to create a new driver.

1 In Designer, open your project.

2 In the Modeler, right-click the driver set where you want to create the driver, then click **New** > **Driver**.

3 Select an appropriate JDBC Base Database Package, such as Oracle Base, then click **Next**.

4 Select the optional features to install for the JDBC driver, then click **Next**.

All options are selected by default. The options are:

Entitlements Support: These packages contain the policies that provision the user accounts on the connected database. For more information, see the *OpenText™ Identity Manager Entitlements Guide (https://www.netiq.com/documentation/identity-manager-4.10/entitlements/data/identity-manager-entitlements.html)*.

Data Collection: These packages contain the policies that enable the driver to collect data for reports. If you are using Identity Reporting, ensure that this option is selected. For more information, see the *Administrator Guide to Identity Reporting (https://www.netiq.com/documentation/identity-manager-4.10/report_setup/data/bookinfo.html)*.

Account Tracking: These packages contain the policies that enable account tracking information for reports. If you are using Identity Reporting, ensure that this option is selected. For more information, see the *Administrator Guide to Identity Reporting (https://www.netiq.com/documentation/identity-manager-4.10/report_setup/data/bookinfo.html)*.

Synchronization Mode: These packages contain the GCVs and sample policies. If you choose the direct/indirect synchronization mode, ensure that you don't change this setting on the driver creation and configuration pages.

By default, the **Show only Applicable packages versions** option is selected.

IMPORTANT: The JDBC packages provide examples of the core functions of the JDBC driver. These examples help you customize the driver for your environment. You can add new policies and settings to the driver to meet your business requirements. The final implementation can be packaged and deployed in Identity Manager.

---

5 (Conditional) If there are package dependencies for the packages you selected to install for this driver, you must install them to install the selected package. Click **OK** to install the package dependency listed.

6 (Conditional) If more than one type of package dependency must be installed, you are presented with separate configuration pages for each package. Continue to click **OK** to install any additional package dependencies.

7 (Conditional) The Common Settings page is displayed only if the Common Settings package is installed as a dependency. On the Install Common Settings page, specify the common settings for User and Group containers:

User Container: Select the Identity Vault container where the user accounts will be added in the Identity Vault. This value becomes the default for all drivers in the driver set.

Group Container: Select the Identity Vault container where the groups will be added in the Identity Vault. This value becomes the default for all drivers in the driver set.

8 Click **Next**.

When all dependencies are installed, the components must be configured.

9 On the Driver Information page, specify a name for the driver that is unique within the driver set, then click **Next**.

10 On the Application Authentication page, fill in the following information for the connected database:

Version: Specify the version of the connected database.

Synchronization Model: Specify the mode of data synchronization based on the selected package.

Data Flow: Specify whether the authoritative source of data is the database, Identity Manager, or bidirectional.

IMPORTANT: Ensure that you don't change the setting for **Synchronization Model** and **Data Flow** options that you selected earlier in the Package Configuration Wizard.

JDBC Implementation: Specify the database connection details.

Connection Information: Specify the database information for the driver to use to connect to the database, such as the IP address, port, and type of the database.

If the database type is selected as SID, you need to enter the Oracle SID value. If the database type is selected as CDB, you need to enter the Oracle Service Name.

Authentication ID: Specify the authentication ID for the connected database.

Password: Specify the password for the driver to connect to the database.

For more information, see "JDBC Driver Settings" on page 50.

11 Click **Next**.

12 Fill in the following fields for Remote Loader information:

Connect To Remote Loader: Select **Yes** or **No** to determine if the driver will use the Remote Loader. For more information, see "Configuring Identity Manager Drivers to Work with the Remote Loader" in the *OpenText™ Identity Manager CE 24.4 (v4.10) Driver Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/bktitle.html)*.

If you select **No**, skip to Step 13. If you select **Yes**, use the following information to complete the configuration of the Remote Loader:

Host Name: Specify the IP address or DNS name of the server where the Remote Loader is installed and running.

Port (Connection): Specify the port number for this driver object. Each driver object connects to the Remote Loader on a separate port. The default value is 8090.

Remote Loader Password: Specify a password to control access to the Remote Loader. It must be the same password that is specified as the Remote Loader password on the Remote Loader.

Driver Password: Specify a password for the driver object to authenticate to the Identity Manager server. It must be the same password that is specified as the Driver Object password on the Remote Loader.

13 Click **Next**.

14 (Conditional) This page is displayed only if you selected to install the Data Collection and Account Tracking groups of packages. On the JDBC Managed System Information page, fill in the following fields to define your connected database application:

Name: Specify a descriptive name for the connected database application. The name is displayed in reports.

Description: Specify a brief description for the connected database application. The description is displayed in reports.

Location: Specify the physical location of the connected database application. The location is displayed in reports.

Vendor: Specify the vendor of the connected database application. This information is displayed in reports.

Version: Specify the version of the connected database application. The version is displayed in reports.

15 Click **Next**.

16 (Conditional) This page is displayed only if you selected to install the Managed System packages and the Account Tracking packages. On the Install JDBC Managed System Information page, fill in the following fields to define the classification of the connected database application:

Classification: Select the classification of the connected database application. This information is displayed in the reports. Your options are:

- Mission-Critical
- Vital
- Not-Critical
- Other

  If you select **Other**, you must specify a custom classification for the JDBC system.

Environment: Select the type of environment the connected database application provides. The options are:

- Development
- Test
- Staging
- Production
- Other

  If you select **Other**, you must specify a custom classification for the database application.

17 Click **Next**.

18 (Conditional) This page is displayed only if you selected to install the Data Collection and Account Tracking groups of packages. Fill in the following fields to define the ownership of the connected database application:

Business Owner: Select a user object in the Identity Vault that is the business owner of the database application. This can only be a user object, not a role, group, or container.

Application Owner: Select a user object in the Identity Vault that is the application owner of the database application. This can only be a user object, not a role, group, or container.

19 Click **Next**.

20 On the Entitlements Information page, specify a name for the **Account Entitlement Value** field, then click **Next**.

21 (Conditional) This page is displayed only if you selected to install the Account Tracking groups of packages. On the Account Tracking page, fill in the following fields:

Connected Database: Specify the connected database application.

Synchronization Model: Specify the mode of data synchronization.

NOTE: Ensure that you don't change the setting that you selected earlier in the Package Configuration Wizard. If you change it after installing the package in a driver object, make sure that you change the **SyncModel** in the Publication Mode GCV.

Object Class: This field is populated based on your selection in the **Synchronization Model**. Specify the table or view in the connected database for which account tracking is enabled. By default, the value is **usr**.

Realm: Specify the name of the realm that uniquely identifies the location of user accounts in the connected database. For example, `mysql.indirect.usr`, where MySQL is the database name with the indirect data synchronization model, and `user` is the table or view in the connected database for which account tracking is enabled.

22 Click **Next**.

23 Review the summary of tasks that will be completed to create the driver, then click **Finish**.

24 After you have installed the driver object, you must change the configuration for your environment. Proceed to "Configuring the Driver Object" on page 48.

Configuring the Driver Object

After importing the packages and creating the driver object, you need to configure the driver to make it operational. There are many settings that can help you customize and optimize the driver. Although it is important for you to understand all of the settings, your first priority should be to configure the driver parameters located on the Driver Configuration page. For information about the driver parameters, see Chapter 6, "Configuring the JDBC Driver," on page 53. After completing the configuration tasks, continue with the next section, Deploying the Driver Object.

NOTE: If the connected system is MS SQL Server database and if you have chosen a direct **Synchronization Model** option, ensure that you change the **Key-Gen-Method** option to **Subscriber Generated** in the Subscriber channel.

Deploying the Driver Object

After the driver object is created in Designer, it must be deployed into the Identity Vault.

1 In Designer, open your project.

2 In the Modeler, right-click the driver icon or the driver line, then select **Live > Deploy**.

3 If you are authenticated to the Identity Vault, skip to Step 5; otherwise, specify the following information:

Host: Specify the IP address or DNS name of the server hosting the Identity Vault.

Username: Specify the DN of the user object used to authenticate to the Identity Vault.

Password: Specify the user's password.

4 Click **OK**.

5 Read the deployment summary, then click **Deploy**.

6 Read the successful message, then click **OK**.

7 Click **Define Security Equivalence** to assign rights to the driver object:

The driver object requires rights to objects within the Identity Vault. The Admin user object is most often used to supply these rights. However, you might want to create a DriversUser (for example) and assign security equivalence to that user. For receiving events from the Identity Vault, ensure that driver object's **Security Equals DN** has the following rights in the Identity Vault:

Entry Rights: The rights to create entries in the Identity Vault.

Attributes Rights: The rights to modify the attributes in the Identity Vault.

7a Click **Add**, then browse to and select the object with the correct rights.

7b Click **OK** twice.

For more information about defining a Security Equivalent User in objects for drivers in the Identity Vault, see "Establishing a Security Equivalent User (https://www.netiq.com/documentation/identity-manager-4.10/security/data/establishing-security-equivalent-user-in-identity-manager.html)" in the *OpenText™ Identity Manager Security Guide (https://www.netiq.com/documentation/identity-manager-4.10/security/data/identity-manager-security-guide.html)*.

8 Click **Exclude Administrative Roles** to exclude users that should not be synchronized:

You should exclude any administrative User objects (for example, Admin and DriversUser) from synchronization.

8a Click **Add**, then browse to and select the user object you want to exclude.

8b Click **OK**.

8c Repeat Step 8a and Step 8b for each object you want to exclude.

8d Click **OK**.

9 Click **OK**.

Starting the Driver Object

When a driver is created, it is stopped by default. To make the driver work, you must start the driver. Identity Manager is an event-driven system, so after the driver is started, it won't do anything until an event occurs. You can use Identity Console or dxevent commands to start the driver.

To start the driver:

1. If you are using the Remote Loader with the driver, make sure the Remote Loader driver instance is running.
2. In Designer, open your project.
3. In the Modeler, right-click the driver icon or the driver line, then select **Live > Start Driver**.
4. Continue with "Activating the Driver" on page 49.

Activating the Driver

The Identity Manager driver for JDBC is part of the Identity Manager Integration Module for Database.

This integration module requires a separate activation. After purchasing the integration module, you will receive activation details in your NetIQ Customer Center.

If you create a new JDBC driver in a driver set that already includes an activated driver from this integration module, the new driver inherits the activation from the driver set.

If you create the driver in a driver set that has not been previously activated with this integration module, the driver will run in the evaluation mode for 90 days. You must activate the driver with this integration module during the evaluation period; otherwise, the driver will be disabled.

If driver activation has expired, the trace displays an error message indicating that you need to reactivate the driver to use it. For information on activation, refer to Activating Identity Manager (https://www.netiq.com/documentation/identity-manager-4.10/idm_overview_planning/data/activating-identity-manager.html) in the *OpenText™ Identity Manager CE 24.4 (v4.10) Overview and Planning Guide (https://www.netiq.com/documentation/identity-manager-4.10/idm_overview_planning/data/identity-manager-overview-and-planning.html)*.

JDBC Driver Settings

You can change the driver options to align with your connected database. Table 5-1 summarizes the JDBC driver settings.

Table 5-1 *JDBC Driver Settings*

Setting	Description
Driver name	The name of the driver that you want to display in the driver set.
Target database	The database that the driver connects to.
Driver is local/remote	Whether the driver runs locally or remotely on a Remote Loader.
Synchronization model	Whether the driver uses views to synchronize directly to existing tables of arbitrary structure or synchronize to intermediate staging tables of a particular structure.
Third-party JDBC implementation	The class of the third-party JDBC file specific to the database being synchronized with Identity Manager. For more information, see "JDBC Driver Class Names" on page 184.
Data flow	Whether the authoritative source of data is the database, Identity Manager, or bidirectional (both the database and Identity Manager).
Database host IP address	The IP address of the database host.
Database port	The port that the driver shim uses to communicate with the database. If you don't provide a port number, the Driver Configuration Wizard provides a default port number for the database that you selected at install time.
User container DN	The distinguished name (complete context) of the container where the database users are published. For example: `data\company\users`.
Group container DN	The distinguished name (complete context) of the container where the database groups are published. For example: `data\company\groups`.

Setting	Description
Publication mode	Whether publication is triggered (default) or triggerless.

6 Configuring the JDBC Driver

- "Smart Configuration" on page 53
- "Configuration Parameters" on page 56
- "Driver Parameters" on page 57
- "Subscription Parameters" on page 82
- "Publication Parameters" on page 90
- "Trace Levels" on page 100
- "Configuring Third-Party JDBC Drivers" on page 101
- "Configuring jTDS Support for the JDBC Driver" on page 101

Smart Configuration

The JDBC driver can recognize the supported set of third-party JDBC drivers and databases. Also, the driver can dynamically and automatically configure the majority of driver compatibility parameters so you don't need to understand and explicitly set such parameters.

These features are implemented via the following four types of XML descriptor files, which describe a third-party JDBC driver or database to the JDBC driver.

- Third-party JDBC driver
- Third-party JDBC driver import
- Database
- Database import

In addition to predefined descriptor files, you can create custom descriptor files for a database or third-party JDBC driver.

- "Specifying Custom Descriptor Files" on page 54
- "Reserved Filenames for Descriptor Files" on page 54
- "Import Descriptor Files" on page 54
- "Descriptor File Locations" on page 54
- "Precedence" on page 55
- "Custom Descriptor Best Practices" on page 55
- "Descriptor File DTDs" on page 55

Specifying Custom Descriptor Files

You can force the driver to use a custom descriptor file for a database or third-party JDBC driver. To specify a custom database descriptor file, see "Database Descriptor Filename" on page 72. To specify a custom third-party driver descriptor file, see "JDBC Driver Descriptor Filename" on page 72. This is useful when multiple descriptor files exist for the same database or third-party JDBC driver. For the custom descriptor file to take effect, set the driver parameter as the jdbc-driver-descriptor.

Reserved Filenames for Descriptor Files

Descriptor filenames that ship with the driver begin with the underscore character (_). Such filenames are reserved to ensure that descriptor files that ship with the driver do not conflict with custom descriptor files. Obviously, custom descriptor filenames must not begin with the underscore character.

Import Descriptor Files

Import descriptor files allow multiple, nonimport descriptor files to share content. This functionality reduces the size of nonimport descriptor files, minimizes the need for repetition of content, and increases maintainability. Import files cannot be imported across major types. That is, JDBC driver descriptors cannot import database imports, and database descriptors cannot import JDBC driver imports.

Furthermore, custom nonimport descriptors cannot import reserved descriptor imports. For example, if a custom third-party JDBC driver descriptor file named `custom.xml` tries to import a reserved third-party JDBC driver descriptor named `_reserved.xml`, an error is issued. These limitations accomplish the following:

- Ensure that no dependencies exist between reserved and custom import files
- Allow extension of existing reserved descriptor files in later versions of the driver

Descriptor File Locations

Descriptor files must be located in a `.jar` file whose name begins with the prefix "jdbc" (case-insensitive) and resides in the runtime classpath.

The following table identifies where to place descriptors within a descriptor `.jar` file:

Table 6-1 Where to Place Descriptors

Descriptor Type	Directory Path
Third-party JDBC driver	`com/novell/nds/dirxml/driver/jdbc/db/descriptor/driver`
Third-party JDBC driver import	`com/novell/nds/dirxml/driver/jdbc/db/descriptor/driver/import`
Database	`com/novell/nds/dirxml/driver/jdbc/db/descriptor/db`
Database import	`com/novell/nds/dirxml/driver/jdbc/db/descriptor/db/import`

Reserved descriptor files are located in the `JDBCConfig.jar` file. To ensure that these reserved files are not overwritten when the JDBC driver is updated, place custom descriptors in a different `.jar` file.

Precedence

Parameters explicitly specified through a management console, such as Identity Console, always have precedence over parameters specified through descriptor files. Descriptor file parameters only take effect when a parameter is not set through the management console.

Parameters and other information specified in a nonimportable descriptor file always have precedence over information specified in descriptor import files. If a parameter or other information is duplicated within a descriptor file, the first instance of the parameter or information takes precedence over subsequent instances.

Among import files, precedence is determined by import order. Import files declared earlier in the import list take precedence over those that follow.

Custom Descriptor Best Practices

- Do not begin custom descriptor files name with the underscore (_) character.
- Place custom descriptor files in a jar file other than `JDBCConfig.jar`, and begin the filename with the prefix “jdbc” (case-insensitive).
- Do not use custom descriptors to import reserved import files (filenames that begin with the underscore character).

Descriptor File DTDs

The following sections contain DTDs for all descriptor file types. These DTDs can help you construct custom descriptor files.

Table 6-2 Where to Find Descriptor DTDs

Descriptor Type	Appendix
Third-party JDBC driver	Appendix H, “Third-Party JDBC Driver Descriptor DTD,” on page 235
Third-party JDBC driver import	Appendix I, “Third-Party JDBC Driver Descriptor Import DTD,” on page 237
Database	Appendix J, “Database Descriptor DTD,” on page 239
Database import	Appendix K, “Database Descriptor Import DTD,” on page 241

Configuration Parameters

- “Viewing Driver Parameters” on page 56
- “Deprecated Parameters” on page 56
- “Authentication Parameters” on page 57

Viewing Driver Parameters

1. In Identity Console, click the **IDM Administration** tile.
2. On the Driver Dashboard, locate the driver icon, then click the upper right corner of the driver icon to display the driver’s properties page.

Deprecated Parameters

The following parameters have been deprecated since version 1.6:

Table 6-3 Deprecated Parameters

Tag Name	Justification
connection-tester-class	The driver now dynamically creates a connection tester class at runtime, based upon information in XML descriptor files. This parameter is still operable, to ensure backwards compatibility. Its continued use, however, is discouraged.
connection-test-stmt	The driver now dynamically creates a connection tester class at runtime, based upon information in XML descriptor files. This parameter is still operable, to ensure backwards compatibility. Its continued use, however, is discouraged.
reconnect-interval	The reconnect interval is now fixed at 30 seconds on both channels.

Authentication Parameters

After you import the driver, provide authentication information for the target database.

- "Authentication ID" on page 57
- "Authentication Context" on page 57
- "Application Password" on page 57

Authentication ID

An Authentication ID is the name of the driver's database user/login account.The installation SQL script for each database provides information on the database privileges required for this account to authenticate to a supported database. The scripts are located in the *install-dir*\DirXMLUtilities\jdbc\sql*abbreviated-database-name*\install directory.

This value can be referenced in the Connection Properties parameter value via the token {$username}. See "Connection Properties" on page 70.

The default value for the sample configuration is idm.

Authentication Context

The authentication context is the JDBC URL of the target database.

URL format and content are proprietary. They differ among third-party JDBC drivers. However, they have some similarities in content. Each URL, whatever the format, usually includes an IP address or DNS name, port number, and a database identifier. For the exact syntax and the content requirements of your driver, consult your third-party driver documentation.

For a list of JDBC URL syntaxes for supported third-party drivers, see "JDBC URL Syntaxes" on page 183.

IMPORTANT: Changing anything in this value other than URL properties forces a resynchronization of all objects when triggerless publication is used.

Application Password

An application password is the password for the driver's database user/login account. The default value for the sample driver configuration is novell.

This value can be referenced in the Connection Properties parameter value via the token {$password}. See "Connection Properties" on page 70.

Driver Parameters

The Driver Parameters section lets you configure the driver-specific parameters. When you change driver parameters, you tune driver behavior to align with your network environment. The following table summarizes all driver-level parameters and their properties:

Table 6-4 Driver Parameters and Properties

Display Name	Tag Name	Sample Value	Default Value	Required
Third-Party JDBC Driver Class Name	jdbc-class	oracle.jdbc.driver.OracleDriver	(none)	yes
Preserve white space in SQL statements?	preserve-sqlwhitespace	yes	no	yes
Synchronization Filter	sync-filter	schema (include by schema membership)	(none)	no
Time Syntax	time-syntax	1 (integer)	1 (integer)	no
State directory	state-dir	. (current directory)	. (current directory)	no
Use Minimal Number of Connections?	use-single-connection	0 (no)	(dynamic[3])	no
Connection Initialization Statements	connection-init	USE idm	(none)	no
Connection Properties	connection-properties	USER={$username}; PASSWORD={$password}	(dynamic[3])	no
JDBC Driver Descriptor Filename	jdbc-driver-descriptor	ora_client_thin.xml	(none)	no
Database Descriptor Filename	database-descriptor	ora_11i.xml	(none)	no
Enable Referential Attribute Support?	enable-refs	1 (yes)	1 (yes)	no
Enable Meta-Identifier Support?	enable-meta-identifiers	1 (yes)	1 (yes)	no
Use Manual Transactions?	use-manual-transactions	1 (yes)	(dynamic[2])	no
Transaction Isolation Level	transaction-isolation-level	read committed	(dynamic[3])	no
Reuse Statements?	reuse-statements	1 (reuse)	(dynamic[3])	no
Number of Returned Result Sets	handle-stmt-results	one	(dynamic[3])	no
Enable Statement-Level Locking?	enable-locking	1 (yes)	0 (no)	no
Force Username Case	force-username-case	upper (to uppercase)	(none)	no
Left Outer Join Operator	left-outer-join-operator	(+)	(dynamic[3])	no

Display Name	Tag Name	Sample Value	Default Value	Required
Retrieve Minimal Metadata	minimal-metadata	0 (no)	(dynamic[3])	no
Function Return Method	function-return-method	result set	(dynamic[3])	no
Supports Schemas in Metadata Retrieval?	supports-schemas-in-metadata-retrieval	1 (yes)	(dynamic[3])	no
Sort Column Names By	column-position-comparator	com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator (hexadecimal value)	(dynamic[3])	no
Schema Name	sync-schema	indirect	(none)	yes[1]
Include Filter Expression	include-table-filter	IDM_.*	(none)	no
Exclude Filter Expression	exclude-table-filter	BIN\$.{22}==\$0	(none)	no
Table/View Names	sync-tables	usr	(none)	yes[1]
Lock Statement Generator Class	lock-generator-class	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator	(dynamic[3])	no

[1] One of these mutually exclusive parameters must be present if the Synchronization Filter parameter is not present. See "Synchronization Filter" on page 64. [2] This default is derived dynamically at runtime from descriptor files and database metadata. [3] This default is derived dynamically from descriptor files at runtime.

Driver parameters fall into the following subcategories:

- "Uncategorized Parameters" on page 59
- "Database Scoping Parameters" on page 64
- "Connectivity Parameters" on page 68
- "Compatibility Parameters" on page 71

Uncategorized Parameters

- "Third-Party JDBC Driver Class Name" on page 60
- "Time Syntax" on page 60
- "State Directory" on page 62

Third-Party JDBC Driver Class Name

This parameter is the fully-qualified Java class name of your third-party JDBC driver.

The following table lists the properties of this parameter:

Table 6-5 *Third-Party JDBC Driver Class Name: Properties*

Property	Value
Tag Name	jdbc-class
Required?	yes
Case-Sensitive?	yes
Sample Value	oracle.jdbc.driver.OracleDriver
Default Value	(none)

For a list of supported third-party JDBC driver classnames, see "JDBC Driver Class Names" on page 184.

Time Syntax

The Time Syntax parameter specifies the format of time-related data types that the driver returns. The format can be any of the following options:

- "Return Database Time, Date, and Timestamp Values as 32-Bit Integers" on page 60
- "Return Database Time, Date, and Timestamp Values as Canonical Strings" on page 61
- "Return database Time, Date, and Timestamp Values in their Java String Representation as Returned by the Method toString():java.lang.String" on page 61

Return Database Time, Date, and Timestamp Values as 32-Bit Integers

This is the default.

eDirectory Time and Timestamp syntaxes are composed of unsigned, 32-bit integers that express the number of whole seconds that have elapsed since 12:00 a.m., January 1st, 1970 UTC. The maximum range of this data type is approximately 136 years. When interpreted as unsigned integers (as originally intended), these syntaxes are capable of expressing dates and times to the second in the range of 1970 to 2106. When interpreted as a signed integer, these syntaxes are capable of expressing dates and times to the second in the range of 1901 to 2038.

This option has two problems:

- Identity Vault Time and Timestamp syntaxes cannot express as large a date range as database Date or Timestamp syntaxes.
- Identity Vault Time and Timestamp syntaxes are granular to the second. Database Timestamp syntaxes are often granular to the nanosecond.

The second and third options overcome these two limitations.

Map the database Time, Date, and Timestamp values to eDirectory attributes of type Time or Timestamp.

Return Database Time, Date, and Timestamp Values as Canonical Strings

The following table shows abstract database data types and their corresponding canonical string representations:

Table 6-6 *Database Types and Canonical String Representations*

JDBC Data Type	Canonical String Format1
java.sql.Time	HHMMSS
java.sql.Date	CCYYMMDD
java.sql.Timestamp	CCYYMMDDHHMMSSNNNNNNNNN

C = century, Y = year, M = month D = day, H = hour, M= minute, S = second, N = nano

These fixed-length formats collate in chronological order on any platform in any locale. Even though the precision of nanoseconds varies by database, the length of Timestamps does not.

Map the database Time, Date, and Timestamp values to attributes of type Numeric String.

Return database Time, Date, and Timestamp Values in their Java String Representation as Returned by the Method toString():java.lang.String

The following table shows abstract database data types and their corresponding Java String representations:

Table 6-7 *Database Types and Java String Formats*

JDBC Data Type	Java String Format1
java.sql.Time	hh:mm:ss
java.sql.Date	yyyy-mm-dd
java.sql.Timestamp	yyyy-mm-dd hh:mm:ss.fffffffff

y= year, m= month, d= day, h= hour, m= minute, s= second, f= nano

These fixed-length formats collate in chronological order on any platform in any locale. The precision of nanoseconds, and therefore the length of Timestamps, varies by database.

Map the database Time, Date, and Timestamp values to attributes of type Case Ignore/Case Exact String.

The following table lists the properties of the Time Syntax parameter:

Table 6-8 Time Syntax: Properties

Property	Value
Tag Name	time-syntax
Required?	no
Default Value	1 (integer)
Legal Values	1 (integer) 2 (canonical string) 3 (java string)
Schema-Dependent?	True

State Directory

The State Directory parameter specifies where a driver instance should store state data. The state data is currently used for both triggered and triggerless publication. For more information, see Triggered Publication Parameters and Triggerless Publication Parameters. The state data might be used to store additional state information in the future.

Each driver instance can have a maximum of four state files with a unique file format, such as:

- `jdbc_<driver instance guid>` - Triggerless Publication (only used for Triggerless configured drivers)
- `jdbc_<driver instance guid>_0` - Subscriber Query
- `jdbc_<driver instance guid>_1` - Publisher Query
- `jdbc_<driver instance guid>_2` - Subscriber Service Query

For example, `jdbc_bd2a3dd5-d571-4171-a195-28869577b87e_0`

Defunct state files (those belonging to deleted drivers) in the state directory are deleted each time a driver instance with the same state directory is started.

Changes That Can Force Triggerless Publisher Resynchronization

If you delete state files, the triggerless publisher will build new state files by resynchronizing. If you move the JDBC driver without moving the state files, the triggerless publisher builds new state files by resynchronizing. Changing to and from the Remote Loader is a move. Therefore, if you move the JDBC driver using triggerless publication and want to prevent a full resynchronization, also move the `jdbc_<GUID>` file in the state directory.

When you move your driver to a new server, the process includes creating a new driver object on the new server. The newly created driver object will have its own GUID. Therefore, copying the state files of the old driver to the new server does not work even if the driver versions are same (or have the same state file extensions). In order to prevent resynchronization, ensure that **Triggerless Publication Legal Values 2 (process future changes only)** option is set in the driver properties.

If more than two files exist in the specified state directory, you must look up the GUID to know which files belong to the driver instance being moved. To identify a driver instance's state files, you can use DSTrace. For convenience, the Identity Manager engine traces each driver's GUID in DSTrace on startup.

If no value is provided for the state directory parameter, or the value is a period (.), the state directory is the current directory. The current directory depends upon the following:

- The platform that the driver is running on
- Whether the driver is running locally or remotely

When a process is started, a default directory in the file system is assigned to it. The default directory is the current directory. If you don't supply a value, the default State Directory is the current directory (the one that the process is running in).

Table 6-9 *Default Directories*

Platform or Environment	Default Directory
Linux and Solaris, for the Remote Loader	`/opt/novell/dirxml`
Linux and Solaris, for Identity Manager (local; not on the Remote Loader)	`/var/opt/novell/eDirectory/data/dib/`
Windows, for the Remote Loader	`novell\remoteloader`
Windows, for Identity Manager (local; not on the Remote Loader)	`c:\novell\nds\dibfiles`

The current directory might be different for a custom installation.

No data is lost when resynchronization occurs, although additional data might remain. For example, because deletes are not captured, users that were deleted in the database during the move will not be disabled/deleted (depending upon the policy).

Moving the driver is not to be undertaken whimsically. As a rule of thumb, don't move the driver unless you must do so.

Properties

The following table lists the properties of the State Directory parameter:

Table 6-10 *State Directory: Properties*

Property	Value
Tag Name	state-dir
Required?	no
Case-Sensitive?	platform-dependent
Sample Value	c:\novell\nds\DIBFiles
Default Value	. (current directory)

Database Scoping Parameters

- "Synchronization Filter" on page 64
- "Schema Name" on page 66
- "Include Filter Expression" on page 67
- "Exclude Filter Expression" on page 67
- "Table/View Names" on page 68

Synchronization Filter

The Synchronization Filter parameter determines which database objects, such as tables and views, are members of the synchronization schema (the set of tables/views visible to the driver at runtime). With the addition of this parameter, the driver can now run in two modes: schema-aware or schema-unaware.

Schema-Unaware Mode

When the Synchronization Filter parameter is present and set to empty (exclude all tables/views), the driver is schema-unaware. It does not retrieve table/view metadata on startup. Therefore, no metadata methods are required. See Appendix F, "java.sql.DatabaseMetaData Methods," on page 227.

When it is schema-unaware, the synchronization schema can be empty. Both the Schema Name and Sync Tables/Views parameters are completely ignored. Neither is required. Both can be absent, present, valued or valueless. See "Schema Name" on page 66 and "Table/View Names" on page 68.

In schema-unaware mode, the driver acts as a passthrough agent for embedded SQL. In this state, standard XDS events (for example, Add, Modify, and Delete) are ignored. See Chapter 12, "Embedded SQL Statements in XDS Events," on page 141. Also, triggered or triggerless publication no longer work.

Schema-Aware Mode

When the Synchronization Filter parameter is not present or set to a value other than empty (exclude all tables/views), the driver is schema-aware. It retrieves table/view metadata on a limited number of tables/views to facilitate data synchronization. You can cache metadata on all tables/views owned by a single database user (include by schema membership), or cache metadata on an explicit list of table/view names (include by table/view name). When schema-aware, the driver retrieves database table/view metadata on startup. For a list of required metadata methods, see Appendix F, "java.sql.DatabaseMetaData Methods," on page 227.

When schema-aware, parameter Schema Name or Table/View Names must be present and have a value. Because these two parameters are mutually exclusive, only one parameter can have a value. See "Schema Name" on page 66 and "Table/View Names" on page 68.

The following table lists the parameters that require the driver to be schema-aware. When the driver is schema-unaware, these parameters do not have any effect on driver behavior.

Table 6-11 Schema-Dependent Parameters

Parameter
Lock Statement Generator Class
Enable Referential Attribute Support?
Enable Meta-Identifier Support?
Left Outer Join Operator
Retrieve Minimal Metadata
Supports Schemas in Metadata Retrieval?
Sort Column Names By
Disable Statement-Level Locking
Check Update Counts?
Add Default Values on Insert?
Generation/Retrieval Method (Table-Global)
Retrieval Timing (Table-Global)
Retrieval Timing
Disable Publisher?
Disable Statement-Level Locking?
Publication Mode
Enable Future Event Processing?
Event Log Table Name
Delete Processed Rows?
Allow Loopback?
Startup Option
Polling Interval (In Seconds)
Publication Time of Day
Post Polling Statements
Batch Size

The following table lists the properties of the Synchronization Filter parameter:

Table 6-12 Synchronization Filter: Properties

Property	Value
Tag Name	sync-filter
Required?	no
Case-Sensitive?	no
Sample Value	list
Legal Values	empty (exclude all tables/views), schema (include by schema membership), list (include by table/view name)
Default Value:	(none)

Schema Name

The Schema Name parameter identifies the database schema being synchronized. A database schema is analogous to the name of the owner of the tables or views being synchronized. For example, to synchronize two tables, `usr` and `grp`, each belonging to database user `idm`, you enter `idm` as this parameter's value. For more information about the Schema Name use cases, see "Schema Name Use Cases" on page 215.

When using this parameter instead of Table/View Names, names of database objects are implicitly schema-qualified by the driver. As such, parameters referencing stored procedure, function, or table names do not need to be schema-qualified unless they reside in a schema other than the one specified here. In particular, Method and Timing (Table-Local) and Event Log Table Name are affected. See "Table/View Names" on page 68, "Method and Timing (Table-Local)" on page 86, and "Event Log Table Name" on page 94.

The following table lists the properties of the Schema Name parameter:

Table 6-13 Schema Name: Properties

Property	Value
Tag Name	sync-schema
Required?	yes
Case-Sensitive?	See "Undelimited Identifier Case Sensitivity" on page 171.
Sample Value	indirect
Default Value:	(none)

When the Schema Name parameter is used without the Synchronization Filter parameter, the Table/View Names parameter must be left empty or omitted from a configuration. See "Synchronization Filter" on page 64 and "Table/View Names" on page 68.

Changing the value of the Schema Name parameter forces a re-sync of all objects when trigger-less publication is used.

Include Filter Expression

The Include Filter Expression parameter is only operative when the Schema Name parameter is used. See "Schema Name" on page 66.

The following table lists the properties of the Include Filter Expression parameter:

Table 6-14 *Include Filter Expression: Properties*

Property	Value
Tag Name	include-table-filter
Required?	no
Case-Sensitive?	yes
Sample Value	idm_.* (all table/view names starting with "idm_")
Default Value	(none)
Legal Values	(any legal Java regular expression)

Exclude Filter Expression

This parameter is only operative when the Schema Name parameter is used. See "Schema Name" on page 66.

The following table lists the properties of the Exclude Filter Expression parameter:

Table 6-15 *Exclude Filter Expression: Properties*

Property	Value
Tag Name	exclude-table-filter
Required?	no
Case-Sensitive?	yes
Sample Value	bin.* (all table/view names starting with "bin")
Default Value	(none)
Legal Values	(any legal Java regular expression)

Table/View Names

The Table/View Names parameter allows you to create a logical database schema by listing the names of the logical database classes to synchronize. Logical database class names are the names of parent tables and views. It is incorrect to list child table names.

This parameter is particularly useful for synchronizing with databases that do not support the concept of schema, such as MySQL, or when a database schema contains a large number of tables or views of which only a few are of interest. Reducing the number of table/view definitions cached by the driver can shorten startup time as well as reduce runtime memory utilization.

When using this parameter instead of Schema Name, you probably need to schema-qualify other parameters that reference stored procedure, function, or table names. In particular, the Method and Timing (Table-Local) and Event Log Table Name parameters are affected. See "Schema Name" on page 66, "Method and Timing (Table-Local)" on page 86 and "Event Log Table Name" on page 94.

The following table lists the properties of the Table/View Names parameter:

Table 6-16 *Table/View Names: Properties*

Property	Value
Tag Name	sync-tables
Required?	yes
Case-Sensitive?	See "Undelimited Identifier Case Sensitivity" on page 171.
Delimiters	semicolon, white space, comma
Sample Value	indirect.usr; indirect.grp
Default Value	(none)

When this parameter is used without the Synchronization Filter parameter, the Schema Name parameter must be left empty or omitted from a configuration. See "Synchronization Filter" on page 64 and "Schema Name" on page 66.

Changing anything in the Table/View Name parameter other than URL properties forces a resynchronization of all objects when triggerless publication is used.

Connectivity Parameters

- "Use Minimal Number of Connections?" on page 69
- "Connection Initialization Statements" on page 69
- "Connection Properties" on page 70

Use Minimal Number of Connections?

The Use Minimal Number of Connections? parameter specifies whether the driver should use two instead of three database connections.

By default, the driver uses three connections: one for subscription, and two for publication. The Publisher channel uses one of its two connections to query for events and the other to facilitate query-back operations.

When this parameter is set to Boolean True, the number of required database connections is reduced to two. One connection is shared between the Subscriber and Publisher channels. It is used to process subscription and publication query-back events. The other is used to query for publication events.

In previous versions, the driver was able to support bidirectional synchronization by using a single connection. The publication algorithm was redesigned to increase performance, enable support for future event processing, and to overcome limitations of the previous algorithm at the expense of requiring an additional connection.

Table 6-17 *Use Minimal Number of Connections?: Properties*

Property	Value
Tag Name	use-single-connection
Required?	no
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	False

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean False.

Setting this parameter to Boolean True reduces performance.

Connection Initialization Statements

The Connection Initialization Statements parameter specifies what SQL statements, if any, should be executed immediately after connecting to the target database. Connection initialization statements are useful for changing database contexts and setting session properties. These statements are executed each time the driver, irrespective of channel, connects or reconnects to the target database.

The following table lists the properties of this parameter:

Table 6-18 Connection Initialization Statements: Properties

Property	Value
Tag Name	connection-init
Required?	no
Case-Sensitive?	See "Undelimited Identifier Case Sensitivity" on page 171.
Delimiters	semicolon
Sample Value	USE idm; SET CHAINED OFF
Default Value	(none)
Schema-Dependent	False

Connection Properties

The Connection Properties parameter specifies authentication properties. This parameter is useful for specifying properties that cannot be set via the JDBC URL specified in the Authentication Context parameter. See "Authentication Context" on page 57.

The primary purpose of this parameter is to enable encrypted transport for third-party JDBC drivers. For a list of relevant connection properties, see "Sybase Adaptive Server Enterprise JConnect JDBC Driver" on page 196 and "Oracle Thin Client JDBC Driver" on page 192.

Connection properties are specified as key-value pairs. The key is specified as the value to the left of the "=" character. The value is the value to the right of the "=" character. You can specify multiple key-value pairs, but each pair must be delimited by the ";" character.

When you use the Connection Properties parameter, authentication information can be passed via the JDBC URL specified in the Authentication Context parameter or here. See "Authentication Context" on page 57.

If specified as connection properties, value tokens can be used as placeholders for the database username specified in the Authentication ID parameter and the password specified in the Application Password parameter. See "Authentication ID" on page 57 and "Application Password" on page 57. For username, the token is `{$username}`. For password, the token is `{$password}`.

The following table lists the properties of this parameter:

Table 6-19 Connection Properties: Properties

Property	Value
Tag Name	connection-properties
Required?	no
Case-Sensitive?	third-party JDBC driver-dependent
Delimiters	semicolon
Sample Value	user={$username}; password={$password}; oracle.jdbc.defaultNChar=true
Default Value	(none)
Schema-Dependent	False

Each connection property is a key value pair. You can add any number of connection properties in the driver configuration. You must append username and password (`user={$username}; password={$password}`) to the additional connection properties. Separate each connection property using delimiters.

Compatibility Parameters

- "JDBC Driver Descriptor Filename" on page 72
- "Database Descriptor Filename" on page 72
- "Use Manual Transactions?" on page 73
- "Transaction Isolation Level" on page 73
- "Reuse Statements?" on page 74
- "Number of Returned Result Sets" on page 75
- "Enable Statement-Level Locking?" on page 76
- "Lock Statement Generator Class" on page 76
- "Enable Referential Attribute Support?" on page 77
- "Enable Meta-Identifier Support?" on page 77
- "Force Username Case" on page 78
- "Left Outer Join Operator" on page 78
- "Retrieve Minimal Metadata" on page 79
- "Function Return Method" on page 79
- "Supports Schemas in Metadata Retrieval?" on page 80
- "Sort Column Names By" on page 80
- "Preserve white space in SQL statements?" on page 81

JDBC Driver Descriptor Filename

The JDBC Driver Descriptor Filename parameter specifies the third-party JDBC descriptor file to use. Descriptor file names must not be prefixed with the underscore character (for example, `_mysql_jdriver.xml`) because such filenames are reserved. Place descriptor files in a jar file beginning with the case-insensitive prefix 'jdbc' (for example `jdbcCustomConfig.jar`) in the jar file's `com/novell/nds/dirxml/driver/jdbc/db/descriptor/db` directory.

The following table lists the properties of this parameter:

Table 6-20 *JDBC Driver Descriptor Filename: Properties*

Property	Value
Tag Name	jdbc-driver-descriptor
Required?	no
Case-Sensitive?	platform-dependent
Sample Value	my_custom_jdbc_driver_descriptor.xml
Default Value	(none)
Schema-Dependent	False

Database Descriptor Filename

The Database Descriptor Filename parameter specifies the database descriptor file to use. Do not use the underscore character in prefixes to Descriptor filenames (for example, `_mysql.xml`). Such names are reserved. Place Descriptor files in a `jar` file beginning with the case-insensitive prefix “jdbc” (for example, `JDBCCustomConfig.jar`). Also, place Descriptor files in the jar file’s `com/novell/nds/dirxml/driver/jdbc/db/descriptor/db` directory.

The following table lists the properties of this parameter:

Table 6-21 *Database Descriptor Filename: Properties*

Property	Value
Tag Name	jdbc-driver-descriptor
Required?	no
Case-Sensitive?	platform-dependent
Sample Value	my_custom_database_descriptor.xml
Default Value	(none)
Schema-Dependent	False

Use Manual Transactions?

The Use Manual Transactions? parameter specifies whether to use manual or user-defined transactions.

This parameter is primarily used to enable interoperability with MySQL MyISAM table types, which do not support transactions.

When set to Boolean True, the driver uses manual transactions. When set to Boolean False, each statement executed by the driver is executed autonomously (automatically).

The following table lists the properties of this parameter:

Table 6-22 *Use Manual Transactions?: Properties*

Property	Value
Tag Name	use-manual-transactions
Required?	no
Case-Sensitive?	no
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	False

The Default Value property is derived dynamically from descriptor files and database metadata at runtime.

To ensure data integrity, set this parameter to Boolean True whenever possible.

Transaction Isolation Level

The Transaction Isolation Level parameter sets the transaction isolation level for connections that the driver uses. Six values exist:

- `unsupported`
- `none`
- `read uncommitted`
- `read committed`
- `repeatable read`
- `serializable`

Five of the values correspond to the public constants defined in the java.sql Interface Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html).

Because some third-party drivers do not support setting a connection's transaction isolation level to none, the driver also supports the additional non-standardized value of `unsupported`. PostgreSQL online documentation (http://www.postgresql.org/docs/current/static/transaction-iso.html) has one of the better, concise primers on what each isolation level actually means.

IMPORTANT: The list of supported isolation levels varies by database. For a list of supported transaction isolation levels for supported databases, see "Supported Transaction Isolation Levels" on page 172.

We recommend using a transaction isolation level of `read committed` because it is the minimum isolation level that prevents the driver from seeing uncommitted changes (dirty reads).

The following table lists the properties of this parameter:

Table 6-23 *Transaction Isolation Level: Properties*

Property	Value
Tag Name	transaction-isolation-level
Required?	no
Case-Sensitive?	no
Default Value	(dynamic)
Legal Values	unsupported, none, read uncommitted, read committed, repeatable read, serializable
Schema-Dependent	False

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is read committed.

Reuse Statements?

The Reuse Statements? parameter specifies whether one or more java.sql.Statement items are active at a time on a given connection. See java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html).

This parameter is primarily used to enable interoperability with "Microsoft JDBC Driver for SQL Server" on page 188

When set to Boolean True, the driver allocates a Java SQL Statement once and then reuses it. When set to Boolean False, the driver allocates/deallocates statement objects each time they are used, ensuring that no more than one statement is active at a time on a given connection.

The following table lists the properties of this parameter:

Table 6-24 Reuse Statements?: Properties

Property	Value
Tag Name	reuse-statements
Required?	no
Case-Sensitive?	no
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	False

The Default Vault is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean True.

Setting this parameter to Boolean False degrades performance.

Number of Returned Result Sets

The Number of Returned Result Sets parameter specifies how many java.sql.Result objects can be returned from an arbitrary SQL statement. See java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html).

This parameter is primarily used to avoid infinite loop conditions in "Oracle Thin Client JDBC Driver" on page 192 when evaluating the results of arbitrary SQL statements.

The following table lists the properties of this parameter:

Table 6-25 Number of Returned Result Sets: Properties

Property	Value
Tag Name	handle-stmt-results
Required?	no
Sample Value	one
Default Value	(dynamic)
Legal Values	none, no (none) single, one (one) multiple, many, yes (multiple)
Schema-Dependent	False

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is multiple, many, or yes.

Enable Statement-Level Locking?

The Enable Statement-Level Locking? parameter specifies whether the driver explicitly locks database resources before executing SQL statements.

The following table lists the properties of this parameter:

Table 6-26 *Enable Statement-Level Locking?: Properties*

Property	Value
Tag Name	enable-locking
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Lock Statement Generator Class

The Lock Statement Generator Class parameter specifies which DBLockStatementGenerator implementation to use to generate the SQL statements necessary to explicitly lock database resources for a pending SQL statement. Information on the DBLockStatementGenerator interface is in the Java documents that ship with the driver.

The following table lists the properties of this parameter:

Table 6-27 *Lock Statement Generator Class: Properties*

Property	Value
Tag Name	lock-generator-class
Required?	no
Sample Value	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Th Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is com.novell.nds.dirxml.driver.jdbc.db.lock.DBLockGenerator.

Enable Referential Attribute Support?

The Enable Referential Attribute Support? parameter toggles whether the driver recognizes foreign key constraints between logical database classes. These are used to denote containment. Foreign key constraints between parent and child tables within a logical database class are unaffected.

When set to Boolean True, foreign key columns are interpreted as referential. When set to Boolean False, foreign key columns are interpreted as non-referential.

The primary purpose of this parameter is to ensure backward compatibility with the 1.0 version of the driver. For 1.0 compatibility, set this parameter to Boolean False.

The following table lists the properties of this parameter:

Table 6-28 *Enable Referential Attribute Support?: Properties*

Property	Value
Tag Name	enable-refs
Required?	no
Default Value	1 (yes)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Enable Meta-Identifier Support?

The Enable Meta-Identifier Support? parameter toggles whether the driver interprets view column name prefixes such as pk_ and fk_ strictly as metadata. When interpreted as metadata, such prefixes are not considered part of the view column name.

For example, when meta-identifier support is enabled, column pk_idu has an effective column name of idu, prohibiting the existence of another column with the same effective name in the same view. When meta-identifier support is disabled, column pk_idu has the effective column name of pk_idu, allowing the existence of another column named idu. Furthermore, when meta-identifier support is enabled, a view with a primary key named pk_idu would conflict with a table having a primary key column named idu. When meta-identifier support is disabled, they would not conflict.

When set to Boolean True, view column prefixes are interpreted as metadata. When set to Boolean False, view column name prefixes are interpreted as part of the column name proper.

The primary purpose of this parameter is to ensure backward compatibility with the 1.5 version of the driver. For 1.5 compatibility, set this parameter to Boolean False.

The following table lists the properties of this parameter:

Table 6-29 Enable Meta-Identifier Support?: Properties

Property	Value
Tag Name	enable-meta-identifiers
Required?	no
Default Value	1 (yes)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Force Username Case

The Force Username Case parameter changes the case of the driver's username used to authenticate to the target database.

The primary purpose of this parameter is to enable interoperability with the Informix JDBC Driver when used against ANSI-compliant databases. See "Informix JDBC Driver" on page 186.

The following table lists the properties of this parameter:

Table 6-30 Force Username Case: Properties

Property	Value
Tag Name	force-username-case
Required?	no
Default Value	(don't force)
Legal Values	lower (to lowercase), mixed (to mixed case), upper (to uppercase), don't force (default)
Schema-Dependent	False

Left Outer Join Operator

The Left Outer Join Operator parameter specifies the left outer join operator used in the triggerless publication query. It might be used for other purposes in the future.

The following table lists the properties of this parameter:

Table 6-31 Left Outer Join Operator: Properties

Property	Value
Tag Name	left-outer-join-operator
Required?	no
Default Value	(dynamic)
Legal Values	*= (+) LEFT OUTER JOIN
Schema-Dependent	True

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is LEFT OUTER JOIN.

Retrieve Minimal Metadata

When set to Boolean True, the driver calls only required metadata methods. When set to Boolean False, the driver calls required and optional metadata methods. For a list of required and optional metadata methods, refer to Appendix F, "java.sql.DatabaseMetaData Methods," on page 227. Optional metadata methods are required for multivalue and referential attribute synchronization.

Table 6-32 Retrieve Minimal Metadata: Properties

Property	Value
Tag Name	minimal-metadata
Required?	no
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean False.

Setting this value to Boolean True improves startup time and third-party JDBC driver compatibility at the expense of functionality.

Function Return Method

The Function Return Method parameter specifies how data is retrieved from database functions.

The primary purpose of this parameter is to enable interoperability with the Informix JDBC driver. See "Informix JDBC Driver" on page 186.

When set to `result set`, function results are retrieved through a result set. When set to `return value`, the function result is retrieved as a single, scalar return value.

Table 6-33 Function Return Method: Properties

Property	Value
Tag Name	function-return-method
Required?	no
Default Value	(dynamic)
Legal Values	result set, return value (scalar return value)
Schema-Dependent	False

The Default Value property is derived dynamically from descriptor files at runtime.

Supports Schemas in Metadata Retrieval?

The Supports Schemas in Metadata Retrieval? parameter specifies whether schema names should be used when retrieving database metadata.

The primary purpose of this parameter is to enable interoperability with the Informix JDBC Driver when used against ANSI-compliant databases. See "Informix JDBC Driver" on page 186.

When set to Boolean True, schema names are used. When set to Boolean False, they are not.

Table 6-34 Supports Schemas in Metadata Retrieval?: Properties

Property	Value
Tag Name	supports-schemas-in-metadata-retrieval
Required?	no
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	False

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is Boolean True.

Sort Column Names By

The Sort Column Names By parameter specifies how column position is to be determined for legacy databases that do not support sorting by column names.

The primary purpose of this parameter is to enable interoperability with legacy databases, such as DB2/AS400.

Sorting columns names by hexadecimal value ensures that if a driver instance is relocated to a different server, it continues to function without modification. Sorting column names by platform or locale string collation order is more intuitive, but might require configuration changes if a driver instance is relocated to a different server. In particular, log table column order and compound

column name order might change. In the case of the latter, Schema-Mapping policies and object association values might need to be updated. In the case of the former, log table columns might have to be renamed.

It is also possible to specify any fully-qualified Java class name as long as the following occur:

- The Java class name implements the java.util.Comparator (http://java.sun.com/j2se/1.5.0/docs/api/java/util/Comparator.html) interface.
- The Java class name accepts java.lang.String (http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html) arguments.
- The class is in the runtime classpath.

Table 6-35 *Sort Column Names By: Properties*

Property	Value
Tag Name	column-position-comparator
Required?	no
Default Value	(dynamic)
Legal Values	com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator (hexadecimal value), com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringComparator (string collation order), (any java.util.Comparator that accepts java.lang.String arguments)
Schema-Dependent	True

The Default Value property is derived dynamically from descriptor files at runtime. Otherwise, the default value is com.novell.nds.dirxml.driver.jdbc.util.config.comp.StringByteComparator.

IMPORTANT: After you set this parameter for a given configuration, don't change the parameter.

Preserve white space in SQL statements?

The Preserve white space in SQL statements? parameter specifies whether trailing spaces in the field names of the SQL statements should be removed or not.

The primary purpose of this parameter is to remove the unintentional trailing spaces from the field names.

If the option is set to no, the trailing spaces are removed from the fields. Set the option to yes, if you do not want to remove the trailing whitespaces.

Table 6-36 *Preserve white space in SQL statements?: Properties*

Property	Value
Tag Name	preserve-sql-whitespace
Required?	yes

Property	Value
Default Value	no
Legal Values	yes, no
Schema-Dependent	False

NOTE: When the Preserve white space in SQL statements? parameter is set to no, ensure that you do not add trailing spaces, because extra spaces are removed while formatting.

Subscription Parameters

The following table summarizes Subscriber-level parameters and their properties:

Table 6-37 *Subscriber-Level Parameters and Properties*

Display Name	Tag Name	Sample Value	Default Value	Required
Disable Subscriber?	disable	1 (yes)	0 (no)	no
Generation/Retrieval Method (Table-Global)	key-gen-method	auto	none (subscription event)	
Retrieval Timing (Table-Global)	key-gen-timing	after (after row insertion)	before (before row insertion)	no
Method and Timing (Table-Local)	key-gen	usr("?=indirect.proc_idu()", before)	(none)	no
Disable Statement-Level Locking?	disable-locking	1 (yes)	0 (no)	no
Check Update Counts?	check-update-count	0 (no)	1 (yes)	no
Add Default Values on Insert?	add-default-values-on-view-insert	0 (no)	(dynamic)	no

This default for the Add Default Values on Insert property is derived dynamically from descriptor files at runtime.

Subscription parameters are in two subcategories:

- “Uncategorized Parameters” on page 83
- “Primary Key Parameters” on page 85

Uncategorized Parameters

- "Disable Subscriber?" on page 83
- "Disable Statement-Level Locking?" on page 83
- "Check Update Counts?" on page 84
- "Add Default Values on Insert?" on page 84

Disable Subscriber?

The Disable Subscriber? parameter specifies whether the Subscriber channel is disabled.

When this parameter is set to Boolean True, the Subscriber channel is disabled. When the parameter is set to Boolean False, the Subscriber channel is active.

Table 6-38 *Disable Subscriber?: Properties*

Property	Value
Tag Name	disable
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	False

Disable Statement-Level Locking?

The Disable Statement-Level Locking? parameter specifies whether database resources are explicitly locked on this channel before each SQL statement is executed. This parameter is active only if Enable Statement-Level Locking? is set to Boolean True.

When this parameter is set to Boolean True, database resources are explicitly locked. When this parameter is set to Boolean False, database resources are not explicitly locked.

Table 6-39 *Disable Statement-Level Locking?: Properties*

Property	Value
Tag Name	disable-locking
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Check Update Counts?

The Check Update Counts? parameter specifies whether the Subscriber channel checks to see if a table was actually updated when `INSERT`, `UPDATE`, and `DELETE` statements executed against a table.

When set to Boolean True, update counts are checked. If nothing is updated, an exception is thrown. When set to Boolean False, update counts are ignored.

When statements are redefined in before-trigger logic, set his parameter to Boolean False

When using Microsoft SQL Server, use the default value, because errors in trigger logic (that might roll back a transaction) are not propagated back to the Subscriber channel.

Table 6-40 *Check Update Counts?: Properties*

Property	Value
Tag Name	check-update-count
Required?	no
Default Value	1 (yes)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Add Default Values on Insert?

The Add Default Values on Insert? parameter specifies whether the Subscriber channel provides default values when executing an `INSERT` statement against a view.

The primary purpose of this parameter is to enable interoperability with Microsoft SQL Server 2000. This database requires that view columns constrained `NOT NULL` have a non-`NULL` value in an `INSERT` statement.

When this parameter is set to Boolean True, default values are provided for `INSERT` statements executed against views, and explicit values are not already available. When this parameter is set to Boolean False, default values are not provided.

Table 6-41 *Add Default Values on Insert?: Properties*

Property	Value
Tag Name	add-default-values-on-view-insert
Required?	no
Default Value	(dynamic)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

The Default Value property is derived dynamically from descriptor files at runtime.

Primary Key Parameters

- "Generation/Retrieval Method (Table-Global)" on page 85
- "Retrieval Timing (Table-Global)" on page 86
- "Method and Timing (Table-Local)" on page 86

When processing <add> events, which map to `INSERT` statements, the Subscriber channel uses primary key values to create Identity Manager associations. These parameters specify how and when the Subscriber channel obtains the primary key values necessary to construct association values. How primary key values are obtained is the primary key generation/retrieval method. The retrieval timing indicates when primary key values are retrieved.

The following table identifies the supported methods and timings:

Table 6-42 *Supported Methods and Timings*

Method	Timing: before (row insertion)	Timing: after (row insertion)
None (subscription event)	Y	N[1]
Driver (Subscriber-generated)	Y	Y
Auto (auto-generated/identity column)	N[2]	Y
(stored procedure/function)	Y	Y

[1] The Subscriber channel automatically overrides this timing and changes it to `before`.

[2] The Subscriber channel automatically overrides this timing and changes it to `after`.

Generation/Retrieval Method (Table-Global)

The Generation/Retrieval Method (Table-Global) parameter specifies how primary key values are generated or retrieved for all parent tables and views. The Method and Timing parameter overrides this parameter on a per-table/view basis. See "Method and Timing (Table-Local)" on page 86.

When this parameter is set to none, primary key values are assumed to already exist in the subscription event. When this parameter is set to `driver`, primary key values are generated by one of the following:

- Using a `SELECT (MAX(pk_columname)+1)` statement if retrieval timing is set to `before`
- Using a `SELECT MAX(pk_columname)` statement if retrieval timing is set to `after`

For string column types, the Subscriber channel generates a value by using the return value of System.CurrentTimeMillis(). Other data types are not supported.

When this parameter is set to `auto`, primary key values are retrieved via the `java.sql.Statement.getGeneratedKeys():java.sql.ResultSet` method. The MySQL Connector/J JDBC driver is the only supported third-party JDBC driver that currently implements this method. See "MySQL Connector/J JDBC Driver" on page 191.

Table 6-43 Generation/Retrieval Method (Table-Global): Properties

Property	Value
Tag Name	key-gen-method
Required?	no
Default Value	none (subscription event)
Legal Values	none (subscription event), driver (Subscriber-generated), auto (auto-generated/identity column)
Schema-Dependent	True

Retrieval Timing (Table-Global)

The Retrieval Timing (Table-Global) parameter specifies when the Subscriber channel retrieves primary key values for all parent tables and views. The Method and Timing (Table-Local) parameter overrides this parameter. See "Method and Timing (Table-Local)" on page 86.

When this parameter is set to `before`, primary key values are retrieved before insertion. When this parameter is set to `after`, primary key values are retrieved after insertion.

Table 6-44 Retrieval Timing (Table-Global): Properties

Property	Value
Tag Name	key-gen-timing
Required?	no
Default Value	before (before row insertion)
Legal Values	before (before row insertion), after (after row insertion)
Schema-Dependent	True

Method and Timing (Table-Local)

The Method and Timing (Table-Local) parameter specifies the primary key generation/retrieval method and retrieval timing on a per parent table/view basis. It essentially maps a generation/retrieval method and retrieval timing to a table or view name. The syntax for this parameter mirrors a procedural programming language method call with multiple arguments (such as *method-name*(*argument1*, *argument2*)).

When using the Table/View Names parameter, you probably need to explicitly schema-qualify any tables, views, stored procedures or functions referenced in this parameter's value. When you use the Schema Name parameter, then tables, views, stored procedures, or functions referenced in this

parameter's value are implicitly schema-qualified with that schema name. If tables, views, stored procedures, or functions referenced in this parameter's value are located in a different schema than the implicit schema, they must be schema-qualified.

- "BNF" on page 87
- "Generation or Retrieval Method" on page 87
- "Retrieval Timing" on page 89

BNF

The BNF (Backus Naur Form) notation for this parameter's value is the following:

```
<key-gen> ::= <table-or-view-name> "(" <generation-retrieval-method>,
                          <retrieval-timing> ")" {[<delimiter>] <key-gen>}

<generation-retrieval-method> ::= none | driver | auto |
                                  """ <procedure-signature> """ |
                                  """ <function-signature>  """

<table-or-view-name> ::= <legal-undelimited-database-table-or-view-
                         identifier>

<delimiter> ::= ";" | "," | <white-space>

<procedure-signature> ::= <schema-qualifier> "." <stored-routine-
                          name>"("<argument-list>")"

<function-signature> ::= "?=" <procedure-signature>

<schema-qualifier> ::= <legal-undelimited-database-username-identifier>

<stored-routine-name> ::= <legal-undelimited-database-stored-routine
                          -identifier>

<argument-list> ::= <column-name>{"," <column-name>}

<column-name> ::= <column-from-table-or-view-name-previously-specified>
```

Generation or Retrieval Method

The generation or retrieval method specifies how primary key values are to be generated, if necessary, and retrieved. The possible methods are None, Driver, Auto, and Stored Procedure/ Function:

None: By default, the Subscriber channel assumes that the Identity Vault is the authoritative source of primary key values and that the requisite values are already present in a given <add> event. If this is the case, no primary values need to be generated because they already exist. They only need to be retrieved from the current <add> event. This method is desirable when an eDirectory attribute, such as GUID, is explicitly schema-mapped to a parent table or view's primary key column.

Assuming the existence of a table named `usr` and a view named `view_usr` where the Identity Vault is the authoritative source of primary key values, this parameter's value would be similar to the following:

```
usr(none); view_usr(none)
```

When you use this method, we recommend mapping GUID rather than CN to a parent table or view's primary key column.

Driver: This method assumes that the driver is the authoritative source of primary key values for the specified parent table or view.

When prototyping or in the initial stages of deployment, it is often desirable to have the Subscriber channel generate primary key values before a stored procedure or function is written. You can also use this method against databases that do not support stored procedures or functions. When you use this method in a production environment, however, all SQL statements generated by an <add> event should be contained in a serializable transaction. For additional information, refer to "Transaction Isolation Level" on page 73.

Instead of making all transactions serializable, you can also set individual transaction isolation levels by using embedded SQL attributes. For additional information, refer to "Transaction Isolation Level" on page 148.

For any numeric column types, the Subscriber channel uses the following to generate primary key values:

- A simple `SELECT(MAX+1)`statement for `before` timing
- A `SELECT MAX()`statement for `after` timing

For string column types, the Subscriber channel generates a value by using the return value of System.CurrentTimeMillis(). Other data types are not supported.

Assuming the existence of a table named `usr` and a view named `view_usr`, where the database is the authoritative source of primary key values, this parameter's value would be similar to the following:

```
usr(driver); view_usr(driver)
```

When you use this method, we recommend that you omit primary key columns from Schema Mapping policies and channel filters.

Auto: This method assumes that the database is the authoritative source of primary key values for the specified parent table or view.

Some databases support identity columns that automatically generate primary key values for inserted rows. This method retrieves auto-generated primary key values through the JDBC 3 interface method `java.sql.Statement.getGeneratedKeys():java.sql.ResultSet`. The MySQL Connector/J JDBC driver is the only supported third-party JDBC driver that currently implements this method. See "MySQL Connector/J JDBC Driver" on page 191.

Assuming the existence of a table named `usr` and a view named `view_usr`, where the database is the authoritative source of primary key values, this parameter's value would be similar to the following:

```
usr(auto); view_usr(auto)
```

When you use this method, we recommend that you omit primary key columns from Schema Mapping policies and channel filters.

Stored-Procedure/Function: This method assumes that the stored-procedure / function is the authoritative source of primary key values for the specified parent table or view.

Assuming

- The existence of a table named `usr` with a primary key column named `idu`
- A view named `view_usr` with a primary key values named `pk_idu`
- The existence of a database function `func_last_usr_idu` and stored procedure `sp_last_view_usr_pk_idu` that both return the last generated primary key value for their respective table/view

This parameter's value would be similar to the following:

```
usr("?=func_last_usr_idu()"); view_usr("sp_last_view_usr_pk_idu(pk_idu)")
```

In the previous examples, a parameter is passed to the stored procedure. Parameters can also be passed to functions, but this is not usually necessary. Unlike functions, stored procedures usually return values through parameters. For stored procedures, primary key columns must be passed as `IN OUT` parameters. Non-key columns must be passed as `IN` parameters.

For both stored procedures and functions, parameter order, number and data type must correspond to the order, number and data type of the parameters expected by the procedure or function.

When you use this method, we recommend that you omit primary key columns from Schema Mapping policies and channel filters.

Retrieval Timing

The Retrieval Timing parameter specifies when primary key values are retrieved.

An <add> event always results in at least one `INSERT` statement against a parent table or view. This portion of this parameter specifies when primary key values are to be retrieved relative to the initial `INSERT` statement.

Before: This is the default setting. When this setting is specified, primary key values are retrieved before the initial `INSERT` statement.

This retrieval timing is supported for all generation/retrieval methods except `auto`. Retrieval timing is required for the `none` method.

After: When this setting is specified, primary key values are retrieved after the initial `INSERT` statement.

This retrieval timing is supported for all generation/retrieval methods except `none`. Retrieval timing is required for the `auto` method.

The following examples augment the previous ones by adding retrieval timing information:

```
usr(none, before); view_usr(none, before)
```

```
usr(driver, before); view_usr(driver, after)
```

```
usr(auto, after); view_usr(auto, after)
```

```
usr("?=func_last_usr_idu()", before);
view_usr("sp_last_view_usr_pk_idu(pk_idu)", after)
```

The following table lists the properties of this parameter:

Table 6-45 Retrieval Timing: Properties

Property	Value
Tag Name	key-gen
Required?	no
Case-Sensitive?	See "Undelimited Identifier Case Sensitivity" on page 171.
Sample Value	usr("?=proc_idu()", before)
Default Value	(none)
Legal Values	(any string adhering to the BNF)
Schema-Dependent	True

Publication Parameters

The following table summarizes publisher-level parameters and their properties:

Table 6-46 Publisher-Level Parameters and Properties

Display Name	Tag Name	Sample Value	Default Value	Required
Disable Publisher?	disable	1 (yes)	0 (no)	no
Disable Statement-Level Locking?	disable-locking	1 (yes)	0 (no)	no
Publication Mode	publication-mode	2 (triggerless)	1 (triggered)	yes
Event Log Table Name	log-table	indirect_process	(none)	yes[1]
Delete Processed Rows?	delete-from-log	0 (no)	1 (yes)	no
Allow Loopback?	allow-loopback	1 (yes)	0 (no)	no
Enable Future Event Processing?	handle-future-events	1 (yes)	0 (no)	no
Startup Option	startup-option			no
Polling Interval (In Seconds)	polling-interval	60	10	no[2]
Publication Time of Day	time-of-day	15:30:00	(none)	no[2]
Post Polling Statements	post-poll-stmt	DELETE FROM direct.direct_process	(none)	yes
Batch Size	batch-size	16	1	no
Query Limit (default 10000)	query-limit	1000	10000	no

Display Name	Tag Name	Sample Value	Default Value	Required
Heartbeat Interval (In Minutes)	pub-heartbeat-interval	10	1	no

[1] Required for triggered publication mode. [2] These parameters are mutually exclusive.

Publication parameters fall into four major subcategories:

- "Uncategorized Parameters" on page 91
- "Triggered Publication Parameters" on page 94
- "Triggerless Publication Parameters" on page 96
- "Polling Parameters" on page 97

Uncategorized Parameters

- "Disable Publisher?" on page 91
- "Disable Statement-Level Locking?" on page 92
- "Publication Mode" on page 92
- "Enable Future Event Processing?" on page 93

Disable Publisher?

The Disable Publisher? parameter specifies whether the Publisher channel is disabled. When disabled, the Publisher channel does not query for database events. Unlike the Disable Subscriber? parameter, you can still issue database queries on the Publisher channel to facilitate alternative publication algorithms.

When this parameter is set to Boolean True, the Publisher channel is disabled. When this parameter is set to Boolean False, the Publisher channel is active.

Table 6-47 *Disable Publisher?: Properties*

Property	Value
Tag Name	disable
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Disable Statement-Level Locking?

The Disable Statement-Level Locking? parameter specifies whether database resources should be explicitly locked on this channel before each SQL statement is executed. This parameter is only active if the Enable Statement-Level Locking? parameter is set to Boolean True.

When this parameter is set to Boolean True, database resources are explicitly locked. When this parameter is set to Boolean False, database resources are not explicitly locked.

Table 6-48 *Disable Statement-Level Locking?: Properties*

Property	Value
Tag Name	disable-locking
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Publication Mode

The Publication Mode parameter specifies which publication algorithm is used.

When set to 1 (triggered), the Publisher channel polls the event log table for events. When set to 2 (triggerless), the Publisher channel searches all tables/views in the synchronization schema for changes, and synthesizes events.

The following table lists the properties of this parameter:

Table 6-49 *Publication Mode: Properties*

Property	Value
Tag Name	publication-mode
Required?	yes
Default Value	1 (triggered)
Legal Values	1 (triggered) 2 (triggerless)
Schema-Dependent	True

Enable Future Event Processing?

For triggered publication, Enable Future Event Processing? specifies whether rows in the event log table are ordered and processed by insertion order (the `record_id` column) or chronologically (the `event_time` column).

When this parameter is set to Boolean False, rows in the event log table are published by order of insertion. When this parameter is set to Boolean True, rows in the event log table are published chronologically.

For triggerless publication, Enable Future Event Processing specifies whether database local time is published with each event. This additional information can be used to force a retry of future-dated events. In order for this to work, a column specifying when an event should be processed must be part of each logical database class utilizing this feature and placed in the Publisher filter as a notification-only attribute.

Database local time is published as an attribute on each XDS event (for example, add, modify, delete). The attribute name is `jdbc:database-local-time`, where the `jdbc` namespace prefix is bound to `urn:dirxml:jdbc`. The format is the Java string representation of a java.sql.Timestamp: `yyyy-mm-dd hh:mm:ss.fffffffff`. Depending upon the value of the Time Syntax parameter, the value indicating when an event should be processed can be published as an integer, as a canonical string, or as a Java string. See "Time Syntax" on page 60.

Regardless of the publication syntax, this value can be parsed and compared to the database local time value. The following table maps the time syntax to the appropriate parse method.

Table 6-50 *Mapping Time Syntax to Parse Methods*

Time Syntax	Parse Method
integer	java.sql.Timestamp(long) (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html)
canonical string	com.novell.nds.dirxml.driver.jdbc.db.DSTime(java.lang.String, java.lang.String, java.lang.String, java.lang.String)
java string	java.sql.Timestamp.valueOf(java.lang.String):java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html)

After both time values are in a common Timstamp object representation, they can be compared by using the following methods:

- com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil.before(java.sql.Timestamp, java.sql.Timestamp):boolean
- com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil.after(java.sql.Timestamp, java.sql.Timestamp):boolean

An example policy is provided in Appendix L, "Policy Example: Trigger-less Future Event Processing," on page 243.

When this parameter is set to Boolean True, local database time is published with each event. When this parameter is set to Boolean False, this information is omitted.

The following table lists the properties of this parameter:

Table 6-51 Enable Future Event Processing?: Properties

Property	Value
Tag Name	handle-future-events
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Triggered Publication Parameters

The JDBC driver can use any of four triggered publication parameters.

- "Event Log Table Name" on page 94
- "Delete Processed Rows?" on page 95
- "Allow Loopback?" on page 95

Event Log Table Name

The Event Log Table Name parameter specifies the name of the event log table where publication events are stored.

The table specified here must conform to the definition of Chapter 11, "The Event Log Table," on page 129.

When using "Table/View Names" on page 68, you should explicitly schema-qualify this table name. When you use "Schema Name" on page 66, this table name is implicitly schema-qualified with that schema name. If this table is located in a schema other than the implicit schema, it must be schema-qualified.

The following table lists the properties of this parameter:

Table 6-52 Event Log Table Name: Properties

Property	Value
Tag Name	log-table
Required?	no
Case-Sensitive?	See "Undelimited Identifier Case Sensitivity" on page 171.
Sample Value	eventlog
Default Value	(none)
Schema-Dependent	True

This parameter is required if "Publication Mode" on page 92 is set to 1 (triggered publication).

Delete Processed Rows?

The Delete Processed Rows? parameter specifies whether processed rows are deleted from the event log table.

When this parameter is set to a Boolean True, processed rows are deleted. When this parameter is set to Boolean False, processed row's `status` field values are updated.

To mitigate the performance hit caused when processed rows remain in the event log table, we recommend periodically moving the rows into a history table. Do one of the following:

- Call a clean-up stored procedure via the parameter "Post Polling Statements" on page 98.
- Place a before-delete trigger on the event log table to intercept delete events executed against the event log table and to move deleted rows to a history table before they are deleted from the event log table.

The following table lists the properties of this parameter:

Table 6-53 *Delete Processed Rows?: Properties*

Property	Value
Tag Name	delete-from-log
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Setting this parameter to Boolean False degrades publication performance unless processed rows are periodically removed from the event log table.

Allow Loopback?

The Allow Loopback? parameter specifies whether events caused by the driver's database user account should be published.

When this parameter is set to Boolean True, loopback events are published. When this parameter is set to Boolean False, loopback events are ignored.

The following table lists the properties of this parameter:

Table 6-54 Allow Loopback?: Properties

Property	Value
Tag Name	allow-loopback
Required?	no
Default Value	0 (no)
Legal Values	1, yes, true (yes) 0, no, false (no)
Schema-Dependent	True

Setting this parameter to Boolean True might degrade performance because extraneous events might be published.

Triggerless Publication Parameters

The Startup Option parameter specifies what happens when a triggerless publisher starts.

Table 6-55 Startup Option: Settings and Results

Setting	Result
1	All objects are assumed to have changed and are republished.
2	Past and present changes are ignored.
3	All past and present changes are published.

The following table lists the properties of this parameter:

Table 6-56 Startup Option: Properties

Property	Value
Tag Name	startup-option
Required?	no
Default Value	1 (process all changes)
Legal Values	1 (resync all objects) 2 (process future changes only) 3 (process all changes)
Schema-Dependent	True

The following configuration changes can force a full resynchronization:

- Changing anything in the Authentication Context parameter other than URL properties forces a resynchronization of all objects when triggerless publication is used.
- Changing the value of the Schema Name parameter or the Table/View Names parameter forces a resynchronization of all objects when triggerless publication is used.

- Changing the State Directory parameter value.
- Moving or deleting state files. See "Changes That Can Force Triggerless Publisher Resynchronization" on page 62.
- Changing the table/view structure in the database (in particular, changing the position or type of key columns).

Polling Parameters

- "Polling Interval (In Seconds)" on page 97
- "Publication Time of Day" on page 97
- "Post Polling Statements" on page 98
- "Batch Size" on page 99
- "Query Limit (default 10000)" on page 99
- "Heartbeat Interval (In Minutes)" on page 100

Polling Interval (In Seconds)

The Polling Interval (In Seconds) parameter specifies how many seconds of inactivity elapse between polling cycles.

The following table lists the properties of this parameter:

Table 6-57 *Polling Interval (In Seconds): Properties*

Property	Value
Tag Name	polling-interval
Required?	no
Default Value	10 (seconds)
Legal Values	1-604800 (1 week)
Schema-Dependent	True

We recommend that you set this value to no less than 10 seconds.

Publication Time of Day

The Publication Time of Day parameter specifies at what time, each day, publication begins. Time is understood to mean server local time (the time on the server where the driver is running). You can specify a single time each day, or multiple times.

The following table lists the properties of this parameter:

Table 6-58 Publication Time of Day: Properties

Property	Value
Tag Name	time-of-day
Required?	no
Sample Value (Single time)	13:00:00 (1PM)
Sample Value (Multiple times)	13:00:00 (1 PM), 16:00:00 (4 PM), 20:00:00 (8PM)
Default Value	(none)
Legal Values	hh:mm:ss (h = hour, m = minute, s = second)
Schema-Dependent	True

This parameter overrides the parameter Polling Interval (In Seconds). See "Polling Interval (In Seconds)" on page 97.

Post Polling Statements

The Post Polling Statements parameter specifies the SQL statements that are executed at the end of each active polling cycle. An active polling cycle is one where some publication activity has occurred.

The primary purpose of this parameter is to allow cleanup of the event log table following publication activity.

You should explicitly schema-qualify any database objects (for example, tables, stored procedures, and functions) referenced in these statements.

The following table lists the properties of this parameter:

Table 6-59 Post Polling Statements: Properties

Property	Value
Tag Name	post-poll-stmt
Required?	yes
Case-Sensitive?	See "Undelimited Identifier Case Sensitivity" on page 171.
Delimiters	semicolon
Sample Value	DELETE FROM direct.direct_process
Default Value	(none)
Legal Values	(any set of legal SQL statements)
Schema-Dependent	True

Batch Size

The Batch Size parameter specifies how many events are sent in a single publication document.

Basically, the larger the batch, the better the performance.

- Larger batches necessitate fewer trips across the network in both directions.
- More events in a single document require fewer trips from the Publisher channel to the Identity Manager engine (assuming that query-back events are not being used).
- Larger batches minimize the number of trips from the Publisher channel to the database (assuming that the third-party JDBC driver and database support batch processing).
- Larger batches require fewer commits to state files in the local file system.

  Commits can also be costly.

This parameter defines an upper bound. The Publisher channel might override the specified value under certain conditions. The upper bound of 128 was chosen to minimize the likelihood of overflowing the Java heap and to mitigate delaying termination of the Publisher thread on driver shutdown.

The following table lists the properties of this parameter:

Table 6-60 *Batch Size: Properties*

Property	Value
Tag Name	batch-size
Required?	no
Default Value	1
Legal Values	1 to 128
Schema-Dependent	True

Query Limit (default 10000)

The Query Limit specifies the maximum number of events that should read from connected system per polling cycle. The default is 10000 events.

Table 6-61 *Query Limit (default 10000): Properties*

Property	Value
Tag Name	query limit
Required?	yes
Default Value	10000
Legal Values	0 to 2,147,483,647 (java.lang.Integer.MAX_VALUE)
Schema-Dependent	False

Heartbeat Interval (In Minutes)

The Heartbeat Interval (In Minutes) parameter specifies how many minutes the Publisher channel can be inactive before it sends a heartbeat document. In practice, more than the number of minutes specified can elapse. That is, this parameter defines a lower bound. The Publisher channel sends a heartbeat document only if the Publisher channel has been inactive for the specified number of minutes. Any publication document sent is, in effect, a heartbeat document.

The following table lists the properties of this parameter:

Table 6-62 *Heartbeat Interval (In Minutes): Properties*

Property	Value
Tag Name	pub-heartbeat-interval
Required?	no
Default Value	0
Legal Values	0 to 2,147,483,647 (java.lang.Integer.MAX_VALUE)
Schema-Dependent	False

Trace Levels

To see debugging output from the driver, add a DirXML-DriverTraceLevel attribute value from 1 to 7 on the driver set containing the driver instance. This attribute is commonly confused with the DirXML-XSL TraceLevel attribute. For more information on driver set trace levels, refer to "Viewing Identity Manager Processes (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/b1rc1vm.html)" in the *OpenText™ Identity Manager CE 24.4 (v4.10) Driver Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/bktitle.html)*.

The driver supports the following seven trace levels:

Table 6-63 *Supported Trace Levels*

Level	Description
1	Minimal tracing
2	Database properties
3	Connection status, SQL statements, event log records
4	Verbose output
5	Database resource allocation/deallocation; state file contents
6	JDBC API (invoked methods, passed arguments, returned values, etc.)
7	Third-party driver

Levels 6 and 7 are particularly useful for debugging third-party drivers.

Configuring Third-Party JDBC Drivers

The following guidelines help you configure third-party drivers. For specific configuration instructions, refer to your third-party driver's documentation.

- Use the latest version of the driver.
- Third-party driver behavior might be configurable.

  In many cases, incompatibility issues can be resolved by adjusting the driver's JDBC URL properties.
- When you work with international characters, you often must explicitly specify to third-party drivers the character encoding that the database uses.

  Do this by appending a property string to the end of the driver's JDBC URL.

  Properties usually consist of a property keyword and character encoding value (for example, `jdbc:odbc:mssql;charSet=Big5`). The property keyword might vary among third-party drivers.

  The possible character encoding values are defined by Sun. For more information, refer to Sun's Supported Encoding Web site (http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html).

The following table lists the recommended settings for maximum driver compatibility. These settings are useful when you use an unsupported third-party driver during initial configuration.

Table 6-64 *Recommended Settings for Third-Party JDBC Drivers*

Parameter Name	Compatibility Value
Synchronization filter	empty
Reuse statements?	0 (no)
Use manual transactions?	0 (no)
Use minimal number of connections?	yes
Retrieve minimal metadata?	1 (yes)
Number of returned result sets	one

Configuring jTDS Support for the JDBC Driver

The JDBC driver can be configured to support jTDS classes. The jTDS classes (jTDS jar files) improve the performance of the driver. The following table defines the set of databases and the driver classes that support the jTDS jar files:

Database Name	Class Name	Connection URL	Jar File Name
MS SQL version 2000/ 2005	net.sourceforge.jtds.jdbc.Driver	jdbc:jtds:sqlserver:// <server>:<port1433>;DatabaseName=<database>	jtds-1.2.2.jar
Sybase	net.sourceforge.jtds.jdbc.Driver	jdbc:jtds:sybase:// <server>:<port5000>;DatabaseName=<database>	jtds-1.2.2.jar

Use the latest jTDS jar file available(jtds-1.2.2.jar).

Place the jar file in the specific directory path for the platform being used. For information on placing the jar files, refer to "Supported Third-Party Jar File Placement" on page 185.

7 Upgrading an Existing Driver

The following sections provide information to help you upgrade an existing driver:

- "What's New" on page 103
- "Working with MapDB 3.0.5" on page 103
- "Considerations for Upgrading the Driver With Different Identity Manager and MapDB Versions" on page 104
- "Upgrading the Driver" on page 105

What's New

What's New in Version 4.1.0.1

This version of the driver does not provide any new features.

What's New in Version 4.1.0

This version of the driver provides the following features:

- Identity Manager 4.7 provides support for MapDB 3.0.5. To ensure that your driver works correctly with Identity Manager 4.7 engine, see "Working with MapDB 3.0.5" on page 103.
- The driver now supports the Subscriber Service channel. This channel enables you to separately process the out-of-band queries without interrupting the normal flow of cached events. For example, the Subscriber Service channel can separately process code map refresh, data collection, and queries triggered from dxcmd. This helps to improve the performance of the driver. For more information, see Improving Driver Performance Using Subscriber Service Channel (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/t41hj9uznkpr.html) in the *OpenText™ Identity Manager CE 24.4 (v4.10) Driver Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/bktitle.html)*.

Working with MapDB 3.0.5

NetIQ recommends that you review the following sections before upgrading your driver to work with Identity Manager 4.7 engine:

- "Understanding Identity Manager 4.7 Engine Support for Driver Versions" on page 104
- "Manually Removing the MapDB Cache Files" on page 104

Understanding Identity Manager 4.7 Engine Support for Driver Versions

- Drivers shipped with Identity Manager 4.7 are compatible with Identity Manager 4.7 Engine or Remote Loader. You must perform the following actions to complete the driver upgrade:
  1. Upgrade the Identity Manager Engine.
  2. (Conditional) Upgrade the Remote Loader.
  3. Upgrade the driver.
  4. Manually remove the MapDB state cache files from the Identity Vault's DIB directory. For more information, see "Manually Removing the MapDB Cache Files" on page 104.
- Drivers shipped before Identity Manager 4.7 are not compatible with Identity Manager 4.7 Engine or Remote Loader.
- Drivers shipped with Identity Manager 4.7 are not backward compatible with Identity Manager 4.6.x Engine or Remote Loader.
- Drivers shipped with Identity Manager 4.7 are not backward compatible with Identity Manager 4.5.x Engine or Remote Loader.

Manually Removing the MapDB Cache Files

The Identity Manager engine upgrade process removes the existing MapDB driver work cache files (`dx*`) from the Identity Vault's DIB directory (`/var/opt/novell/eDirectory/data/dib` or `C:\Novell\NDS\DIBFiles`). You must manually remove the existing MapDB state cache files for the driver after upgrading the driver. The MapDB state cache files for the JDBC driver are represented in the following format:

```
jdbc_<driver instance guid>_*
```

where ***** is the name of the state cache file. For example, `jdbc_<driver instance guid>_0` or `jdbc_<driver instance guid>_1.t`

This action ensures that your driver works correctly with Identity Manager 4.7 engine.

Considerations for Upgrading the Driver With Different Identity Manager and MapDB Versions

If the newer driver includes a different version of state file persistence API (MapDB or ZoomDB), then the upgraded driver in a triggerless mode can no longer make use of the state files from the old driver. The below table provides details about the state persistence API versions that can help you determine whether the existing state files will continue to work with the newer driver version.

Identity Manager Version	Persistence API Version	JDBC Driver Version	State File
Identity Manager 4.7	mapdb 3.0.5	4.1.0.1	jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8

Identity Manager Version	Persistence API Version	JDBC Driver Version	State File
Identity Manager 4.6.3	mapdb 1.0.9	4.0.5.0	◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8 ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.p ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.t
Identity Manager 4.6	mapdb 1.0.9	4.0.3.0	◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8 ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.p ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.t **NOTE:** The version of the driver version shipped with Identity Manager 4.6 is 4.0.3.0. However, the version of the driver shipped with Identity Manager 4.5.6 is 4.0.4.1. Therefore, when moving from version 4.5.6 to 4.6, ensure that you upgrade the driver to 4.0.4.1.
Identity Manager 4.5.6	mapdb 1.0.8	4.0.4.1	◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8 ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.p ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.t
Identity Manager 4.5	mapdb 1.0.4	4.0.0.2	◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8 ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.p ◆ jdbc_e53fdaf6-b825-4074-8cf5-f6da3fe525b8.t

NOTE: Prior to Identity Manager 4.x, the extension of driver state files was `<tao number>.db` or `.lg`.

Upgrading the Driver

The driver upgrade process involves upgrading the installed driver packages and updating the existing driver files. These are independent tasks and can be separately planned for a driver. For example, you can update the driver packages and choose not to update the driver files at the same time. However, you are recommended to complete all the update steps within a short amount of time to ensure that the driver has the latest updates.

- "Upgrading the Installed Packages" on page 106
- "Updating the Driver Files" on page 107

Before starting the upgrade process, ensure that you have taken a back-up of the current driver configuration.

Upgrading the Installed Packages

1 Download the latest available packages.

   To configure Designer to automatically read the package updates when a new version of a package is available, click **Windows > Preferences > NetIQ > Package Manager > Online Updates** in Designer. However, if you need to add a custom package to the Package Catalog, you can import the package `.jar` file. For more information about creating custom packages, see Developing Packages (https://www.netiq.com/documentation/identity-manager-4.10/designer_admin/data/packmandevelop.html) in the *OpenText™ Designer for Identity Manager Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/designer_admin/data/bookinfo.html)*.

2 Upgrade the installed packages.

   2a Open the project containing the driver.

   2b Right-click the driver for which you want to upgrade an installed package, then click **Driver > Properties**.

   2c Click **Packages**.

   If there is a newer version of a package, there is check mark displayed in the Upgrades column.

   2d Click **Select Operation** for the package that indicates there is an upgrade available.

   2e From the drop-down list, click **Upgrade**.

   2f Select the version that you want to upgrade to, then click **OK**.

   NOTE: Designer lists all versions available for upgrade.

   2g Click **Apply**.

   2h (Conditional) Fill in the fields with appropriate information to upgrade the package, then click **Next**.

   Depending on which package you selected to upgrade, you must fill in the required information to upgrade the package.

   2i Read the summary of the packages that will be installed, then click **Finish**.

   2j Review the upgraded package, then click **OK** to close the Package Management page.

   For detailed information, see the Upgrading Installed Packages (https://www.netiq.com/documentation/identity-manager-4.10/designer_admin/data/packman.html#packmanupgrade) in the *OpenText™ Designer for Identity Manager Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/designer_admin/data/bookinfo.html)*.

Updating the Driver Files

This section provides general instructions for updating the driver files. For information about updating the driver files to a specific version, search for that driver patch in the Patch Finder Download Page and follow the instructions from the Readme file that accompanies the driver patch release.

To update the driver files:

1. Stop the driver instance by using Identity Console, Designer, or dxcmd by performing one of the following actions:
   - If the driver is running locally, stop the driver instance and the Identity Vault.
   - If the driver is running with a Remote Loader instance, stop the driver and the Remote Loader instance.

   For example, go to a command prompt on Linux and run `ndsmanage stopall`
2. Download the driver patch file to a temporary folder on your server.
3. Extract the contents of the driver patch file.
4. Update the driver files:
   - **Linux:** Open a command prompt and run the following command to upgrade the existing RPM:

     ```
     rpm -U (image-path)/netiq-DXMLRESTJDBC.rpm
     ```
   - **Windows:** Navigate to the `<Extracted Driver Patch File Temporary Location>\windows` folder and copy the `JDBCDriverShim.jar` file to `<IdentityManager installation>\RemoteLoader\lib` folder.
5. (Conditional) If the driver is running locally, start the Identity Vault and the driver instance.

   For example, open a command prompt on Linux and run `ndsmanage startall`
6. (Conditional) If the driver is running with a Remote Loader, start the Remote Loader and the driver instance.

8 Managing the Driver

As you work with the JDBC driver, there are a variety of management tasks you might need to perform, including the following:

- Starting, stopping, and restarting the driver
- Viewing driver version information
- Using Named Passwords to securely store passwords associated with the driver
- Monitoring the driver's health status
- Backing up the driver
- Inspecting the driver's cache files
- Viewing the driver's statistics
- Using the DirXML Command Line utility to perform management tasks through scripts
- Securing the driver and its information
- Synchronizing objects
- Migrating and resynchronizing data
- Activating the driver

Because these tasks, as well as several others, are common to all Identity Manager drivers, they are included in one reference, the *OpenText™ Identity Manager CE 24.4 (v4.10) Driver Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/bktitle.html)*.

9 Schema Mapping

- "High-Level View" on page 111
- "Logical Database Classes" on page 111
- "Indirect Synchronization" on page 111
- "Direct Synchronization" on page 120
- "Synchronizing Primary Key Columns" on page 123
- "Synchronizing Multiple Classes" on page 124
- "Mapping Multivalue Attributes to Single-Value Database Fields" on page 124

High-Level View

The following table shows a high-level view of how the driver maps OpenText Identity Vault objects to database objects.

Table 9-1 *Mapping Identity Vault Objects to Database Objects*

Identity Vault Object	Database Object
Tree	Schema
Class	Table/View
Attribute	Column
Association	Primary Key

Logical Database Classes

A logical database class is the set of tables or the view used to represent an OpenText eDirectory™ class in a database. A logical database class can consist of a single view or one parent table and zero or more child tables.

The name of a logical database class is the name of the parent table or view.

Indirect Synchronization

In an indirect synchronization model, the driver maps the following:

Table 9-2 Mappings in Indirect Synchronization

Identity Vault Object	Database Object
Classes	Tables
Attributes	Columns
1 Class	1 parent table and 0 or more child tables
Single-value attribute	Parent table column
Multivalue attribute	Parent table column (holding delimited values) or Child table column (preferred)

- "Mapping OpenText eDirectory Classes to Logical Database Classes" on page 112
- "Parent Tables" on page 114
- "Parent Table Columns" on page 115
- "Child Tables" on page 115
- "Referential Attributes" on page 117
- "Single-Value Referential Attributes" on page 117
- "Multivalue Referential Attributes" on page 118

Mapping OpenText eDirectory Classes to Logical Database Classes

In the following example, the logical database class `usr` consists of the following:

- One parent table `usr`
- Two child tables: `usr_phone` and `usr_faxno`.

Logical class `usr` is mapped to the OpenText eDirectory class User.

```
CREATE TABLE indirect.usr
(
    idu         INTEGER  NOT NULL,
    fname       VARCHAR2(64),
    lname       CHAR(64),
    pwdminlen   NUMBER(4),
    pwdexptime  DATE,
    disabled    NUMBER(1),
    username    VARCHAR2(64),
    loginame    VARCHAR2(64),
    photo       LONG RAW,
    manager     INTEGER,
    CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
    CONSTRAINT fk_usr_manager FOREIGN KEY (manager)
        REFERENCES indirect.usr(idu)
)

CREATE TABLE indirect.usr_phone
(
    idu      INTEGER       NOT NULL,
    phoneno  VARCHAR2(64)  NOT NULL,
    CONSTRAINT fk_phone_idu FOREIGN KEY (idu)
        REFERENCES indirect.usr(idu)
)

CREATE TABLE indirect.usr_fax
(
    idu    INTEGER       NOT NULL,
    faxno  VARCHAR2(64)  NOT NULL,
    CONSTRAINT fk_fax_idu FOREIGN KEY (idu)
        REFERENCES indirect.usr(idu)
)

<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>Given Name</nds-name>
            <app-name>fname</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Surname</nds-name>
            <app-name>lname</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Password Expiration Time</nds-name>
            <app-name>pwdexptime</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>jpegPhoto</nds-name>
            <app-name>photo</app-name>
        </attr-name>
        <attr-name class-name="User">
```

```
            <nds-name>manager</nds-name>
            <app-name>manager</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Password Minimum Length</nds-name>
            <app-name>pwdminlen</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Facsimile Telephone Number</nds-name>
            <app-name>usr_fax.faxno</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Telephone Number</nds-name>
            <app-name>usr_phone.phoneno</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Login Disabled</nds-name>
            <app-name>disabled</app-name>
        </attr-name>
    </attr-name-map>
</rule>
```

Parent Tables

Parent tables are tables with an explicit primary key constraint that contains one or more columns. In a parent table, an explicit primary key constraint is required so that the driver knows which fields to include in an association value.

```
CREATE TABLE indirect.usr
(
    idu  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu PRIMARY KEY (idu)
)
```

The following table contains sample data for table `indirect.usr`.

idu	fname	lname
1	John	Doe

The resulting association for this row is

`idu=1,table=usr,schema=indirect`

The case of database identifiers in association values is determined dynamically from database metadata at runtime.

Parent Table Columns

Parent table columns can contain only one value. As such, they are ideal for mapping single-value OpenText eDirectory attributes, such as mapping the single-value OpenText eDirectory attribute Password Minimum Length to the single-value parent table column `pwdminlen`.

Parent table columns are implicitly prefixed with the schema name and name of the parent table. It is not necessary to explicitly table-prefix parent table columns. For example, `indirect.usr.fname` is equivalent to `fname` for schema mapping purposes.

```
<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>Given Name</nds-name>
            <app-name>fname</app-name>
        </attr-name>
    </attr-name-map>
</rule>
```

Large binary and string data types should usually be mapped to parent table columns. To map to a child table column, data types must be comparable in SQL statements. Large data types usually cannot be compared in SQL statements.

Large binary and string data types can be mapped to child table columns if the following occur:

- Each `<remove-value>` event on these types is transformed in a policy into a `<remove-all-values>`element
- An `<add-value>` element follows each `<remove-value>` event

Child Tables

A child table is a table that has a foreign key constraint on its parent table's primary key, linking the two tables together. The columns that comprise the child table's foreign key can have different names than the columns in the parent table's primary key.

The following example shows the relationship between parent table `usr` and child tables `usr_phone` and `usr_faxno`:

```
CREATE TABLE indirect.usr
(
    idu  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu  PRIMARY KEY (idu)
)
```

```
CREATE TABLE indirect.usr_phone
(
    idu      INTEGER       NOT NULL,
    phoneno  VARCHAR2(64)  NOT NULL,
    CONSTRAINT fk_phone_idu FOREIGN KEY (idu)         REFERENCES
indirect.usr(idu)
)
```

```
CREATE TABLE indirect.usr_fax
(
    idu    INTEGER       NOT NULL,
    faxno  VARCHAR2(64)  NOT NULL,
    CONSTRAINT fk_fax_idu FOREIGN KEY (idu)         REFERENCES
indirect.usr(idu)
)
```

In a child table, constrain all columns NOT NULL.

The first constrained column in a child table identifies the parent table. In the above example, the constrained column in child table `usr_phone` is `idu`. The only purpose of this column is to relate tables `usr_phone` and `usr`. Because constrained columns do not contain any useful information, omit them from publication triggers and Schema Mapping policies.

The unconstrained column is the column of interest. It represents a single, multivalue attribute. In the above example, the unconstrained columns are phoneno and `faxno`. Because unconstrained columns can hold multiple values, they are ideal for mapping multivalue eDirectory attributes (for example, mapping the multivalue eDirectory attribute Telephone Number to `usr`phone`.`phoneno).

The following table contains sample data for `indirect.usr_phone`.

Table 9-3 *Sample Data*

idu	phoneno
1	111-1111
1	222-2222

Like parent table columns, child table columns are implicitly schema-prefixed. Unlike parent table columns, however, a child table column name must be explicitly prefixed with the child table name (for example, `usr_phone`.phoneno). Otherwise, the driver implicitly interprets column phoneno (the parent table column) as `usr`.phoneno, not the child table column `usr_phone`.phoneno.

```
<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>Facsimile Telephone Number</nds-name>
            <app-name>usr_fax.faxno</app-name>
        </attr-name>
        <attr-name class-name="User">
            <nds-name>Telephone Number</nds-name>
            <app-name>usr_phone.phoneno</app-name>
        </attr-name>
      </attr-name-map>
</rule>
```

Map each multivalue eDirectory attribute to a different child table.

Referential Attributes

You can represent referential containment in the database by using foreign key constraints. Referential attributes are columns within a logical database class that refer to the primary key columns of parent tables in the same logical database class or those of other logical database classes.

Single-Value Referential Attributes

You can relate two parent tables through a single-value parent table column. This column must have a foreign key constraint pointing to the other parent table's primary key. The following example relates a single parent table `usr` to itself:

```
CREATE TABLE indirect.usr
(
    idu      INTEGER  NOT NULL,
    -- ...
    manager  INTEGER,
    CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
    CONSTRAINT fk_usr_manager FOREIGN KEY (manager)         REFERENCES
indirect.usr(idu)
)
```

NOTE: Single-valued referential columns should be nullable.

```
<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>manager</nds-name>
            <app-name>manager</app-name>
        </attr-name>
    </attr-name-map>
</rule>
```

In the above example, each user can have only one manager who himself is a user.

Multivalue Referential Attributes

You can relate two parent tables through a common child table. This child table must have a column constrained by a foreign key pointing to the other parent table's primary key. The following example relates two parent tables `usr` and `grp` through a common child table `member`.

```
CREATE TABLE indirect.usr
(
    idu  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu PRIMARY KEY (idu)
)

CREATE TABLE indirect.grp
(
    idg  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_grp_idg PRIMARY KEY (idg)
)

CREATE TABLE indirect.grp_member
(
    idg  INTEGER  NOT NULL,
    idu  INTEGER  NOT NULL,
    CONSTRAINT fk_member_idg FOREIGN KEY (idg)      REFERENCES
indirect.grp(idg),    CONSTRAINT fk_member_idu FOREIGN KEY (idu)
REFERENCES indirect.usr(idu)
)
```

Constrain all columns in a child table `NOT NULL`.

```
<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
             <nds-name>Group</nds-name>
             <app-name>indirect.grp</app-name>
        </class-name>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="Group">
            <nds-name>Member</nds-name>
            <app-name>grp_member.idu</app-name>
        </attr-name>
    </attr-name-map>
</rule>
```

The first constrained column in a child table determines which logical database class the child table `grp_member` belongs to. In the above example, `grp_member` is considered to be part of logical database class `grp`. `grp_member` is said to be a proper child of `grp`. The second constrained column in a child table is the multivalue referential attribute.

In the following example, the order of the constrained columns has been reversed so `that` `grp_member` is part of class `usr`. To more accurately reflect the relationship, table `grp_member` has been renamed to `usr_mbr_of`.

```
CREATE TABLE indirect.usr
(
    idu  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_usr_idu PRIMARY KEY (idu)
)

CREATE TABLE indirect.grp
(
    idg  INTEGER  NOT NULL,
    -- ...
    CONSTRAINT pk_grp_idg PRIMARY KEY (idg)
)

CREATE TABLE indirect.usr_mbr_of
(
    idu  INTEGER  NOT NULL,
    idg  INTEGER  NOT NULL,
    CONSTRAINT fk_mbr_of_idu FOREIGN KEY (idu)         REFERENCES
indirect.usr(idu) ON DELETE CASCADE,
    CONSTRAINT fk_mbr_of_idg FOREIGN KEY (idg)
        REFERENCES indirect.grp(idg) ON DELETE CASCADE
)
```

```
<rule name="Schema Mapping Rule">
    <attr-name-map>
        <class-name>
             <nds-name>Group</nds-name>
             <app-name>indirect.grp</app-name>
        </class-name>
        <class-name>
            <nds-name>User</nds-name>
            <app-name>indirect.usr</app-name>
        </class-name>
        <attr-name class-name="User">
            <nds-name>Group Membership</nds-name>
            <app-name>usr_mbr_of.idg</app-name>
        </attr-name>
    </attr-name-map>
</rule>
```

In databases that have no awareness of column position (such as DB2/AS400), order is determined by sorting column names by string or hexadecimal value. For additional information, see "Sort Column Names By" on page 80.

In general, it is necessary to synchronize only bidirectional, multivalue, referential attributes as part of one class or the other, not both. If you want to synchronize referential attributes for both classes, construct two child tables, one for each class. For example, if you want to synchronize OpenText eDirectory attributes Group Membership and Member, you need two child tables.

In practice, when you synchronize User and Group classes, we recommend that you synchronize the Group Membership attribute of class User instead of the Member attribute of class Group. Synchronizing the group memberships of a user is usually more efficient than synchronizing all members of a group.

Direct Synchronization

In a direct synchronization model, the driver maps the following:

Table 9-4 *Mappings in Direct Synchronization*

Identity Vault Object	Database Object
Classes	Views
Attributes	View Columns
Class	View
Single-value attribute	View Column
Multivalue attribute	View Column

The update capabilities of views vary between databases. Most databases allow views to be updated when they are comprised of a single base table. (That is, they do not join multiple tables.) If views are read-only, they cannot be used for subscription. Some databases allow update logic to be defined on views in instead-of-triggers, which allow a view to join multiple base tables and still be updateable.

For a list of databases that support instead-of-triggers, see "Database Features" on page 168. Instead-of-trigger logic can be simulated, regardless of database capability using embedded SQL. See "Virtual Triggers" on page 146.

- "View Column Meta-Identifiers" on page 121
- "Primary Key Columns" on page 123
- "Schema Mapping" on page 123

View Column Meta-Identifiers

A view is a logical table. Unlike tables, views do not physically exist in the database. As such, views usually cannot have traditional primary key/foreign key constraints. To simulate these constructs, the JDBC driver embeds constraints and other metadata in view column names. The difference between these constraints and traditional ones is that the former are not enforced at the database level. They are an application-level construct.

For example, to identify to the driver which fields to use when constructing association values, place a primary key constraint on a parent table. The corollary to this for a view is to prefix one or more column names with `pk_` (case-insensitive).

The following table lists the constraint prefixes that can be embedded in view column names.

Table 9-5 *Constraint Prefixes*

Constraint Prefixes (case-insensitive)	Interpretation
pk_	primary key
fk_	foreign key
sv_	single-value
mv_	multivalue

The following example views contain all of these constraint prefixes. These are examples and not the actual samples. Therefore, they should not be used in the driver implementation. The real samples are bundled with the Identity Manager media.

```
CREATE VIEW direct.view_usr
(
    pk_idu,             -- primary key column; implicitly single-valued
    sv_fname,           -- single-valued column
    mv_phoneno,         -- multi-valued column
    fk__idu__manager,   -- self-referential foreign key column; refers
                        --   to primary key column idu in view_usr;
                        --   implicitly single-valued
    fk_mv__idg__mbr_of  -- extra-referential foreign key column; refers
                        --   to primary key column idg in view_grp;
                        --   multi-valued
)
AS
-- ...
```

```
CREATE VIEW direct.view_grp
(
    pk_idg,            -- primary key column; implicitly single-valued
    fk_mv__idu__mbr    -- extra-referential foreign key column; refers
                       --   to primary key column idu in view_usr;
                       --   multi-valued
)
AS
-- ...
```

BNF

The BNF (Backus Naur Form) notation for view column meta-identifiers:

```
<view-column-name> ::= [<meta-info>] <column-name>

<column-name> ::= <legal-unquoted-database-identifier>
<meta-info> ::= <referential> | <non-referential>

<non-referential> ::= [<single-value> | <multiple-value>]

<single-value> ::= "sv_"

<multiple-value> ::= "mv_"

<referential> ::= <primary-key> | <foreign-key>

<primary-key> ::= "pk_" [<single-value>] [<column-group-id>]
                  [<referenced-column-name>]

<column-group-id> ::= <non-negative-integer> "_"

<referenced-column-name> ::= "_" <column-name> "__"

<foreign-key> ::= "fk_" [<non-referential>] [<column-group-id>]
                  <referenced-column-name>
```

Normalized Forms

By default, all view column names are single-valued. Therefore, explicitly specifying the `sv_` prefix in a view column name is redundant. For example, `sv_fname` and `fname` are equivalent forms of the same column name.

Also, primary key column names implicitly refer to themselves. Therefore, it is redundant to specify the referenced column name. For example, `pk_idu` is equivalent to `pk__idu__idu`.

The JDBC driver uses two normalized forms of view meta-identifiers:

- Database native form

  Database native form is the column name as declared in the database. This form is usually much more verbose than schema mapping form, and contains all necessary meta information.

- Schema mapping form

Schema mapping form is returned when the driver returns the application schema. This form is much more concise than database native form because much of the meta information included in database native form is represented in XDS XML and not in the identifier.

The referential prefixes `pk_` and `fk_` are the only meta information preserved in schema mapping form. This limitation ensures backward compatibility.

The following table provides examples of each form:

Table 9-6 *Example Normalized Forms of View Meta-Identifiers*

Database Native Form	Schema Mapping Form
pk_idu	pk_idu
sv_fname	fname
mv_phoneno	phoneno
fk_mv__idg__mbr_of	fk_mbr_of

Equivalent Forms

A view column name without meta information is called its "effective" name, which is similar to a directory object's "effective" rights. For the driver, view column name equivalency is determined without respect to meta information by default. For example, `pk_idu` is equivalent to `idu`, and `fk_mv__idg__mbr_of` is equivalent to `mbr_of`. Any variant form of a view meta column identifier can be passed to the driver at runtime. For backward compatibility reasons, meta information can be treated as part of the effective view column name. See "Enable Meta-Identifier Support?" on page 77.

Primary Key Columns

Primary key column names must be unique among all views in the synchronization schema.

Schema Mapping

Schema mapping conventions for views and view columns are equivalent to that used for parent tables and parent table columns.

Synchronizing Primary Key Columns

When the database is the authoritative source of primary key columns, generally omit the columns from the Publisher and Subscriber filters, Schema Mapping policies, and publication triggers.

When the Identity Vault is the authoritative source of primary key columns, include the columns in the Subscriber filter and Schema Mapping policies, but omit the columns from the Publisher filter and publication triggers. Also, GUID rather than CN is recommended for use as a primary key. CN is a multivalue attribute and can change. GUID has a single value and is static.

Synchronizing Multiple Classes

When synchronizing multiple eDirectory classes, synchronize each class to a different parent table or view. Each logical database class must have a unique primary key column name. The Publisher channel uses this common column name to identify all rows in the event log table pertaining to a single logical database class. For example, both the logical database classes `usr` and `grp` have a unique primary key column name.

```
CREATE TABLE usr
(
    idu   INTEGER         NOT NULL,
    lname VARCHAR2(64)  NOT NULL,
    --...
    CONSTRAINT pk_usr_idu PRIMARY KEY(idu)
);

CREATE TABLE grp
(
    idg  INTEGER  NOT NULL,
    --...
    CONSTRAINT pk_grp_idg PRIMARY KEY(idg)
);
```

Mapping Multivalue Attributes to Single-Value Database Fields

By default, the driver assumes that all eDirectory attributes mapped to parent table columns or view columns have a single value. Because the driver is unaware of the OpenText eDirectory schema, it has no way of knowing whether an eDirectory attribute has a single value or has multiple values. Accordingly, multivalue and single-value attribute mappings are handled identically.

The driver implements the Most Recently Touched (MRT) algorithm with regard to single-value parent table or view columns. An MRT algorithm ensures that the most recently added attribute value or most recently deleted attribute value is stored in the database. The algorithm is adequate if the attribute in question has a single value.

If the attribute has multiple values, the algorithm has some undesirable consequences. When a value is deleted from a multivalue attribute, the database field it is mapped to is set to `NULL` and remains `NULL` until another value is added. The preferred solution to this undesirable behavior is to extend the OpenText eDirectory schema so that only single-value attributes are mapping to parent table or view columns.

Other solutions include the following:

- For indirect synchronization, map each multivalue attribute to its own child table.
- For both direct or indirect synchronization, use a policy to delimit multiple values before inserting them into a table or view column.
- Implement a first or last value per replica policy in style sheets by using methods provided in the `com.novell.nds.indirect.driver.jdbc.util.MappingPolicy` class. Under a first-value-per-replica (FPR) policy, the first attribute value on the OpenText eDirectory replica is always synchronized. Under a last-value-per-replica (LPR) policy, the last attribute value on a

replica is always synchronized. By using global configuration values, you can configure the sample driver configuration to use either FPR or LPR mapping policies. Multivalue to single-value attribute mapping policies are contained in the Subscriber Command Transformation policy container. The sample driver configuration maps the multivalue OpenText eDirectory attributes Given Name and Surname to the single-value columns `fname` and `lname` respectively.

10 Mapping XDS Events to SQL Statements

- "Mapping XDS Events for Indirect Synchronization" on page 127
- "Mapping XDS Events for Direct Synchronization" on page 127

Mapping XDS Events for Indirect Synchronization

The following table summarizes how the Subscriber channel maps XDS events to DML SQL statements for indirect synchronization:

Table 10-1 *Mapping XDS Events for Indirect Synchronization*

XML Event	SQL Equivalent
`<add>`	◆ 0 or more select statements, depending upon the matching policy. ◆ 1 parent table insert statement for all single value `<add-attr>` elements. ◆ 0 or 1 stored procedure/function calls to retrieve primary key values before or after the parent table insert statement. ◆ 1 child table insert statement for each multivalue `<add-attr>` element.
`<modify>`	◆ 1 parent table update statement for each single value `<add-value>` or `<remove-value>` element. ◆ 1 child table insert statement for each multivalue `<add-value>` element. ◆ 1 child table delete statement for each `<remove-value>` element.
`<delete>`	◆ 1 parent table delete statement. ◆ 1 delete statement for each child table.
`<query>`	◆ 1 parent table select statement. ◆ 1 select statement for each childtable.
`<move>` `<rename>` `<modify-password>` `<check-object-password>`	◆ 0 statements unless bound to embedded SQL statements.

Mapping XDS Events for Direct Synchronization

The following table summarizes how the Subscriber channel maps XDS events to DML SQL statements for direct synchronization:

Table 10-2 Mapping XDS Events for Direct Synchronization

XML Event	SQL Equivalent
`<add>`	◆ 0 or more select statements, depending upon the matching policy. ◆ 1 view insert statement for all single value `<add-attr>` element. ◆ 0 or 1 stored procedure/function call to retrieve primary key values before or after the view insert statement. ◆ 1 view insert statement for each multivalue `<add-attr>` element.
`<modify>`	◆ 1 view update statement for each single value `<add-value>` or `<remove-value>` element. ◆ 1 view insert statement for each multivalue `<add-value>` element. ◆ 1 view delete statement for each `<remove-value>` element.
`<delete>`	◆ 1 view delete statement.
`<query>`	◆ 1 view select statement.
`<move>` `<rename>` `<modify-password>` `<check-object-password>`	◆ 0 statements unless bound to embedded SQL statements.

11 The Event Log Table

The event log table stores publication events. This section describes the structure and capabilities of the event log table.

You can customize the name of the event log table and its columns to avoid conflicts with reserved database keywords. The order, number, and data types of its columns, however, are fixed. In databases that are unaware of column position, order is determined by the Sort Column Names By parameter. See "Sort Column Names By" on page 80.

Events in this table can be ordered either by order of insertion (the `record_id` column) or chronologically (the `event_time` column). Ordering events chronologically allows event processing to be delayed. To order publication events chronologically, set the Enable Future Event Processing parameter to Boolean True. See "Enable Future Event Processing?" on page 93.

- "Event Log Columns" on page 129
- "Event Types" on page 132

Event Log Columns

This section describes columns in the event log table. Columns are ordered by position.

- "record_id" on page 129
- "table_key" on page 129
- "status" on page 130
- "event_type" on page 130
- "event_time" on page 131
- "perpetrator" on page 131
- "table_name" on page 131
- "column_name" on page 131
- "old_value" on page 131
- "new_value" on page 132

record_id

The `record_id` column is used to uniquely identify rows in the event log table and order publication events. This column must contain sequential, ascending, positive, unique integer values. Gaps between `record_id` values no longer prematurely end a polling cycle.

table_key

Format values for this column are exactly the same in all triggers for a logical database class. The BNF or Backus Naur Form of this parameter is defined below:

```
<table-key> ::= <unique-row-identifier> {"+"
                <unique-row-identifier>}

<unique-row-identifier> ::= <primary-key-column-name> "=" <value>
```

For example, for the `usr` table referenced throughout this chapter, this column's value might be `idu=1`.

For the `view_usr` view referenced throughout this chapter, this column's value might be `pk_empno=1`.

For a hypothetical compound primary key (one containing multiple columns), this column's value might be *pkey1=value1+pkey2=value2*.

If primary key values placed in the `table_key` field contain any of the special characters {, ; ' + " = \ < >}, where { and } contain the set of special characters, delimit the value with double quotes. You also need to escape the double quote character " as \" and the literal escape character \ as \\ when they are contained inside a pair of double quotes.

For a hypothetical primary key containing special characters, this column's value might be `pkey=", ; ' + \" = \\ < >"`. (Note the double quotes and escaped characters.)

Differences in padding or formatting might result in out-of-order event processing. For performance reasons, remove any unnecessary white space from numeric values. For example, `idu=1` is preferred over `idu= 1`.

status

The `status` column indicates the state of a given row. The following table lists permitted values:

Table 11-1 *Permitted Values for Status Columns*

Character Value	Interpretation
N	new
S	success
W	warning
E	error
F	fatal

To be processed, all rows inserted into the event log table must have a `status` value of N. The remainder of the status characters are used solely by the Publisher channel to designate processed rows. All other characters are reserved for future use.

Status values are case sensitive.

event_type

Values in this column must be between 1 and 8. All other numbers are reserved for future use.

The following table describes each event type:

Table 11-2 Event Types

Event Type	Interpretation
1	insert field
2	update field
3	update field (remove all values)
4	delete row
5	insert row (query-back)
6	update row (query-back)
7	insert field (query-back)
8	update field (query-back)

For additional information on this field, see "Event Types" on page 132.

event_time

This column serves as an alternative ordering column to `record_id`. It contains the effective date of the event. It must not be `NULL`. For this column to become the ordering column, set the Enable Future Event Processing parameter to Boolean True. See "Enable Future Event Processing?" on page 93.

perpetrator

This column identifies the database user who instigated the event. A `NULL` value is interpreted as a user other than the driver user. Rows with a `NULL` value or value not equal to the driver's database username are published. Rows with a value equal to the driver's database username are not published unless the Allow Loopback Publisher parameter is set to Boolean True. See "Allow Loopback?" on page 95.

table_name

The name of the table or view where the event occurred.

column_name

The name of the column that was changed. This column is used only for per-field (1-3, 7-8) event types. Nevertheless, it must always be present in the event log table. If it is missing, the Publisher channel cannot start.

old_value

The field's old value. This column is used only for per-field, non-query-back event types (1-3). Nevertheless, it must always be present in the event log table. If it is missing, the Publisher channel cannot start.

new_value

The field's new value. This column is used only by per-field, non-query-back event types (1-3). Nevertheless, it must always be present in the event log table. If it is missing, the Publisher channel cannot start.

Event Types

The following table describes each event type:

Table 11-3 *Event Types*

Event Type	Interpretation
1	insert field
2	update field
3	update field (remove all values)
4	delete row
5	insert row (query-back)
6	update row (query-back)
7	insert field (query-back)
8	update field (query-back)

Event types are in four major categories. Some categories overlap. The following table describes each category and indicates which event types are members:

Table 11-4 *Event Categories and Types*

Event Category	Event Types
Per-field (attribute)	1, 2, 3, 7, 8
Per-row (object)	4, 5, 6
Non-query-back	1, 2, 3, 4
Query-back	5, 6, 7, 8
Per-field, non-query-back	1, 2, 3
Per-field, query-back	7, 8
Per-row, non-query-back	4
Per-row, query-back	5, 6

In general, a combination of event types from each category yields the best trade-off in terms of space, time, implementation complexity, and performance.

Per-field event types are more granular, require more space, and are more complex to implement than per-row event types. Per-row events are less granular, require less space, and are easier to implement than per-field event types.

Query-back event types use less space but require more time to process than non-query-back event types. Non-query-back event types use more space but require less time to process than query-back event types.

Query-back event types take precedence over their non-query-back counterparts. Non-query-back events are ignored if a query-back event is logged for the same field or object. For example, if an event of type 2 (update-field, non-query-back) and 8 (update-field, query-back) are logged on the same field, the type 2 event is ignored in favor of the type 8 event.

Furthermore, query-back row event types take precedence over query-back field event types. For example, if an event type 8 (update field, query-back) and a event type 6 (update row query-back) are logged on the same object, the type 8 event is ignored in favor of the type 6 event.

Query-back events are ignored by the Publisher if the database object no longer exists. They are dependent upon the database object still being available at processing time. Therefore, logged query-back adds and modifies (event types 5, 6, 7, 8) have no effect once the database object they refer to is deleted.

The following table shows the basic correlation between publication event types and the XDS XML generated by the Publisher channel.

Table 11-5 *Basic Correlation of Publication Event Types*

Event Type	Resulting XDS
insert	<add>
update	<modify>
delete	<delete>

The following example illustrates XML that the Publisher channel generates for events logged on the `usr` table for each possible event type.

```
CREATE TABLE indirect.usr
(
    idu   INTEGER  NOT NULL,
    fname VARCHAR2(64),
    photo LONGRAW,
    --...
    CONSTRAINT pk_usr_idu PRIMARY KEY(idu)
);
```

The following table shows the initial contents of `usr` after a new row has been inserted:

Table 11-6 *An Inserted Row in the `usr` Table*

idu	fname	lname	photo
1	Jack	Frost	0xAAAA

The following table shows the current contents of usr after the row has been updated:

Table 11-7 An Updated Row in the usr Table

idu	fname	lname	photo
1	John	Doe	0xBBBB

Insert Field

The table below shows the contents of the event log table after a new row is inserted into table usr. The value for column photo has been Base64-encoded. The Base64-encoded equivalent of 0xAAAA is qqo=.

Table 11-8 Event Log Table: Insert Field

event_type	table	table_key	column_name	old_value	new_value
1	usr	idu=1	fname	NULL	Jack
1	usr	idu=1	lname	NULL	Frost
1	usr	idu=1	photo	NULL	qqo=

The Publisher channel generates the following XML:

```
<add class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <add-attr attr-name="fname">
        <value type="string">Jack</value>
    </add-attr>
    <add-attr attr-name="lname">
        <value type="string">Frost</value>
    </add-attr>
    <add-attr attr-name="photo">
        <value type="octet">qqo=</value>
    </add-attr>
</add>
```

Update Field

The following table shows the contents of the event log table after the row in table usr has been updated. The values for column photo has been Base64-encoded. The Base64-encoded equivalent of 0xBBBB is u7s=.

Table 11-9 Event Log Table: Update Field

event_type	table	table_key	column_name	old_value	new_value
2	usr	idu=1	fname	Jack	John
2	usr	idu=1	lname	Frost	Doe
2	usr	idu=1	photo	qqo=	u7s=

The Publisher channel generates the following XML:

```
<modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <modify-attr attr-name="fname">
        <remove-value>
            <value type="string">Jack</value>
        </remove-value>
        <add-value>
            <value type="string">John</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="lname">
        <remove-value>
            <value type="string">Frost</value>
        </remove-value>
        <add-value>
            <value type="string">Doe</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="photo">
        <remove-value>
            <value type="octet">qqo=</value>
        </remove-value>
        <add-value>
            <value type="octet">u7s=</value>
        </add-value>
    </modify-attr>
</modify>
```

Update Field (Remove-All-Values)

The following table shows the contents of the event log table after the row in table `usr` has been updated. The value for column `photo` has been Base64-encoded.

Table 11-10 Event Log Table: Update Field (Remove-All-Values

event_type	table	table_key	column_name	old_value	new_value
3	usr	idu=1	fname	Jack	John
3	usr	idu=1	lname	Frost	Doe
3	usr	idu=1	photo	qqo=	u7s=

The Publisher channel generates the following XML:

```
<modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <modify-attr attr-name="fname">
        <remove-all-values/>
        <add-value>
            <value type="string">John</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="lname">
        <remove-all-values/>
        <add-value>
            <value type="string">Doe</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="photo">
        <remove-all-values/>
        <add-value>
            <value type="octet">u7s=</value>
        </add-value>
    </modify-attr>
</modify>
```

Delete Row

The table below shows the contents of the event log table after the row in table `usr` has been deleted.

Table 11-11 *Event Log Table: Delete Row*

event_type	table	table_key	column_name	old_value	new_value
4	usr	idu=1	NULL	NULL	NULL

The Publisher channel generates the following XML:

```
<delete class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
</delete>
```

Insert Row (Query-Back)

The following table shows the contents of the event log table after a new row is inserted into table `usr`.

Table 11-12 *Event Log Table: Insert Row (Query-Back)*

event_type	table	table_key	column_name	old_value	new_value
5	usr	idu=1	NULL	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```
<add class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <add-attr attr-name="fname">
        <value type="string">John</value>
    </add-attr>
    <add-attr attr-name="lname">
        <value type="string">Doe</value>
    </add-attr>
    <add-attr attr-name="photo">
        <value type="octet">u7s=</value>
    </add-attr>
</add>
```

Update Row (Query-Back)

The table below shows the contents of the event log table after the row in table `usr` has been updated.

Table 11-13 *Event Log Table: Update Row (Query-Back)*

event_type	table	table_key	column_name	old_value	new_value
6	usr	idu=1	NULL	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```
<modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <modify-attr attr-name="fname">
        <remove-all-values/>
        <add-value>
            <value type="string">John</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="lname">
        <remove-all-values/>
        <add-value>
            <value type="string">Doe</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="photo">
        <remove-all-values/>
        <add-value>
            <value type="octet">u7s=</value>
        </add-value>
    </modify-attr>
</modify>
```

Insert Field (Query-Back)

The following table shows the contents of the event log table after a new row is inserted into table `usr`. Old and new values are omitted because they are not used.

Table 11-14 *Event Log Table: Insert Field (Query-Back)*

event_type	table	table_key	column_name	old_value	new_value
7	usr	idu=1	fname	NULL	NULL
7	usr	idu=1	lname	NULL	NULL
7	usr	idu=1	photo	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```
<add class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <add-attr attr-name="fname">
        <value type="string">John</value>
    </add-attr>
    <add-attr attr-name="lname">
        <value type="string">Doe</value>
    </add-attr>
    <add-attr attr-name="photo">
        <value type="octet">u7s=</value>
    </add-attr>
</add>
```

Update Field (Query-Back)

The following table shows the contents of the event log table after the row in table `usr` has been updated. Old and new values are omitted because they are not used.

Table 11-15 *Event Log Table: Update Field (Query-Back)*

event_type	table	table_key	column_name	old_value	new_value
8	usr	idu=1	fname	NULL	NULL
8	usr	idu=1	lname	NULL	NULL
8	usr	idu=1	photo	NULL	NULL

The Publisher channel generates the following XML. The values reflect the current contents of table `usr`, not the initial contents.

```
<modify class-name="usr">
    <association>idu=1,table=usr,schema=indirect
    </association>
    <modify-attr attr-name="fname">
        <remove-all-values/>
        <add-value>
            <value type="string">John</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="lname">
        <remove-all-values/>
        <add-value>
            <value type="string">Doe</value>
        </add-value>
    </modify-attr>
    <modify-attr attr-name="photo">
        <remove-all-values/>
        <add-value>
            <value type="octet">u7s=</value>
        </add-value>
    </modify-attr>
</modify>
```

12 Embedded SQL Statements in XDS Events

Embedded SQL allows you to embed SQL statements in XDS-formatted XML documents. You can use embedded SQL statements along with XDS events or use them alone. When embedded SQL statements are used alone, embedded SQL processing does not require that the driver know anything about tables/view in the target database. Therefore, the driver can run in schema-unaware mode. See "Synchronization Filter" on page 64. When using embedded SQL alone, you must establish associations manually. The driver won't establish them for you.

When used in conjunction with XDS events, embedded SQL can act as a virtual database trigger. In the same way that you can install database triggers on a table and cause side effects in a database when certain SQL statements are executed, embedded SQL can cause side effects in a database in response to certain XDS events.

All examples in this section reference the following `indirect.usr` table.

```
CREATE TABLE indirect.usr
(
    idu   INTEGER  NOT NULL,
    fname VARCHAR2(64),
    lname VARCHAR2(64),

    CONSTRAINT pk_usr_idu PRIMARY KEY(idu)
);
```


- "Common Uses of Embedded SQL" on page 142
- "Embedded SQL Basics" on page 142
- "Token Substitution" on page 143
- "Virtual Triggers" on page 146
- "Manual vs. Automatic Transactions" on page 147
- "Transaction Isolation Level" on page 148
- "Statement Type" on page 149
- "SQL Queries" on page 150
- "Data Definition Language (DDL) Statements" on page 152
- "Logical Operations" on page 153
- "Implementing Password Set with Embedded SQL" on page 154
- "Implementing Modify Password with Embedded SQL" on page 155
- "Implementing Check Object Password" on page 155
- "Calling Stored Procedures and Functions" on page 156
- "Best Practices" on page 165

Common Uses of Embedded SQL

You can accomplish the following by embedding SQL in XDS events:

- Create database users or roles.
- Set, check, or modify user passwords.
- Manage database user or role privileges.

For examples of each, consult the `User` DDL Command Transformation style sheet on the Subscriber channel in the example driver configuration.

Embedded SQL Basics

- "Elements" on page 142
- "Namespaces" on page 142
- "Embedded SQL Example" on page 143

Elements

SQL is embedded in XDS events through the `<jdbc:statement>` and `<jdbc:sql>` elements. The `<jdbc:statement>` element can contain one or more `<jdbc:sql>` elements.

Namespaces

The namespace prefix `jdbc` used throughout this section is implicitly bound to the namespace `urn:dirxml:jdbc` when referenced outside of an XML document.

You must use namespace-prefixed embedded SQL elements and attributes. Otherwise, the driver does not recognize them. In all examples in this section, the prefix used is `jdbc`. In practice, the prefix can be whatever you want it to be, as long as it is bound to the namespace value urn:dirxml:jdbc.

The following XML example illustrates how to use and properly namespace-prefix embedded SQL elements. In the following example, the namespace declaration and namespace prefixes are bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr">
         <add-attr name="lname">
             <value>Doe</value>
         </add-attr>
     </add>
     <jdbc:statement>
         <jdbc:sql>UPDATE indirect.usr SET fname = 'John'          </jdbc:sql>
     </jdbc:statement>
</input>
```

Embedded SQL Example

The following XML example illustrates how to use the `<jdbc:statement>` and `<jdbc:sql>` elements and their interpretation. In the following example, embedded SQL elements are bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr">
         <add-attr name="lname">
             <value>Doe</value>
         </add-attr>
     </add>
     <jdbc:statement>
         <jdbc:sql>UPDATE indirect.usr SET fname = 'John'          </
jdbc:sql>
     </jdbc:statement>
</input>
```

Because the Subscriber channel resolves <add> events to one or more `INSERT` statements, the XML shown above resolves to:

```
SET AUTOCOMMIT OFF
INSERT INTO indirect.usr(lname)VALUES('Doe');
COMMIT; --explicit commit
UPDATE indirect.usr SET fname = 'John';
COMMIT; --explicit commit
```

Token Substitution

Rather than require you to parse field values from an association, the Subscriber channel supports token substitution in embedded SQL statements. In the following examples, tokens and the values they reference are bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <modify class-name="usr">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">
            <add-value>
                <value>DoeRaeMe</value>
            </add-value>
        </modify-attr>
    </modify>
    <jdbc:statement>
           <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                     idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

Token placeholders must adhere to the XSLT attribute value template syntax {$*field-name*}. Also, the referenced association element must precede the `<jdbc:statement>` element in the XDS document, or must be present as a child of the `<jdbc:statement>` element. Alternatively, instead of copying the association element as child of the `<jdbc:statement>` element, you could copy the `src-entry-id` of the element containing the association element onto the <jdbc:statement> element. Both approaches are bolded in the following examples:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <modify class-name="usr">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">
            <add-value>
                <value>DoeRaeMe</value>
            </add-value>
        </modify-attr>
    </modify>
    <jdbc:statement>
           <association>idu=1,table=usr,schema=indirect</association>
           <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                     idu = {$idu}</jdbc:sql>
    </jdbc:statement>

</input>
```

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <modify class-name="usr" src-entry-id="0">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">
            <add-value>
                <value>DoeRaeMe</value>
            </add-value>
        </modify-attr>
    </modify>
    <jdbc:statement src-entry-id="0">
           <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                     idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

The {*$field-name*} token must refer to one of the naming RDN attribute names in the association value. The above examples have only one naming attribute: `idu`.

An <add> event is the only event where an association element is not required to precede embedded SQL statements with tokens because the association has not been created yet. Additionally, any embedded SQL statements using tokens must follow, not precede, the <add> event. For example:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr">
          <add-attr name="lname">
               <value>Doe</value>
          </add-attr>
     </add>
     <jdbc:statement>
          <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                    idu = {$idu}</jdbc:sql>
     </jdbc:statement>
</input>
```

To prevent tracing of sensitive information, you can use the {`$$password`} token to refer to the contents of the immediately preceding <`password`> element within the same document. In the following example, the password token and the value it refers to are bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr">
          <password>some password</password>
          <add-attr name="fname">
              <value>John</value>
          </add-attr>
          <add-attr name="lname">
              <value>Doe</value>
          </add-attr>
     </add>
     <jdbc:statement>
          <jdbc:sql>CREATE USER jdoe IDENTIFIED BY
                    {$$password}</jdbc:sql>
     </jdbc:statement>
</input>
```

Furthermore, you can also refer to the driver's database authentication password specified by the Application Password parameter as `{$$$driver-password}`. See "Application Password" on page 57. Named password substitution is not yet supported.

Just as with association elements, the referenced password element must precede the `<jdbc:statement>` element in the XDS document or must be present as a child of the `<jdbc:statement>` element. Alternatively, instead of copying the password element as child of the `<jdbc:statement>` element, you could copy the `src-entry-id` of the element containing the password element onto the `<jdbc:statement>` element. Both approaches are bolded in the following examples:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr">
          <password>some password</password>
          <add-attr name="fname">
              <value>John</value>
          </add-attr>
          <add-attr name="lname">
              <value>Doe</value>
          </add-attr>
     </add>
     <jdbc:statement>
          <password>some password</password>
          <jdbc:sql>CREATE USER jdoe IDENTIFIED BY
                    {$$password}</jdbc:sql>
     </jdbc:statement>
</input>
```

```
<input xmlns:jdbc="urn:dirxml:jdbc">
     <add class-name="usr" src-entry-id="0">
          <password>some password</password>
          <add-attr name="fname">
              <value>John</value>
          </add-attr>
          <add-attr name="lname">
              <value>Doe</value>
          </add-attr>
     </add>
     <jdbc:statement src-entry-id="0">
          <jdbc:sql>CREATE USER jdoe IDENTIFIED BY
                    {$$password}</jdbc:sql>
     </jdbc:statement>
</input>
```

Virtual Triggers

In the same way that database triggers can fire before or after a triggering statement, embedded SQL can be positioned before or after the triggering XDS event. The following examples show how you can embed SQL before or after an XDS event.

Virtual Before Trigger

```
<input xmlns:jdbc"urn:dirxml:jdbc">
     <jdbc:statement>
          <association>idu=1,table=usr,schema=indirect</association>
       <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
idu = {$idu}</jdbc:SQL>
    </jdbc:statement>
    <modify class-name="usr">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">
            <remove-all-values/>
            <add-value>
                <value>Doe</value>
            </add-value>
        </modify-attr>
    </modify>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
COMMIT; --explicit commit
UPDATE indirect.usr SET lname = 'Doe'  WHERE idu = 1;
COMMIT; --explicit commit
```

Virtual After Trigger

```
<input xmlns:jdbc"urn:dirxml:jdbc">
    <modify class-name="usr">
        <association>idu=1,table=usr,schema=indirect</association>
        <modify-attr name="lname">
            <remove-all-values/>
            <add-value>
                <value>Doe</value>
            </add-value>
        </modify-attr>
    </modify>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE         idu
= {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
UPDATE indirect.usr SET lname = 'Doe' WHERE idu = 1;
COMMIT; --explicit commit
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
COMMIT; --explicit commit
```

Manual vs. Automatic Transactions

You can manually group embedded SQL and XDS events by using two custom attributes:

- `jdbc:transaction-type`
- `jdbc:transaction-id`

jdbc:transaction-type

This attribute has two values: `manual` and `auto`. By default, most XDS events of interest (<add>, <modify>, and <delete>) are implicitly set to the manual transaction type. The manual setting enables XDS events to resolve to a transaction consisting of one or more SQL statement.

By default, embedded SQL events are set to auto transaction type because some SQL statements, such as DDL statements, cannot usually be included in a manual transaction. In the following example, the attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr" jdbc:transaction-type="auto">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                  idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT ON
INSERT INTO indirect.usr(lname) VALUES('Doe');
-- implicit commit
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
-- implicit commit
```

jdbc:transaction-id

The Subscriber channel ignores this attribute unless the element's `jdbc:transaction-type` attribute value defaults to or is explicitly set to `manual`. The following XML shows an example of a manual transaction. The attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr" jdbc:transaction-id="0">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
     <jdbc:statement jdbc:transaction-type="manual"
jdbc:transaction-id="0">
           <jdbc:sql>UPDATE indirect.usr SET fname = 'John' WHERE
                     idu = {$idu}</jdbc:sql>
     </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
INSERT INTO indirect.usr(lname) VALUES('Doe');
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
COMMIT; -- explicit commit
```

Transaction Isolation Level

In addition to grouping statements, you can use transactions to preserve the integrity of data in a database. Transactions can lock data to prevent concurrent access or modification. The isolation level of a transaction determines how locks are set. Usually, the default isolation level that the driver uses is sufficient and should not be altered.

The custom attribute `jdbc:isolation-level` allows you to adjust the isolation transaction level if necessary. The java.sql.Connection parameter defines five possible values in the interface. See java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html).

- `none`
- `read uncommitted`
- `read committed`
- `repeatable read`
- `serializable`

The driver's default transaction isolation level is `read committed` unless overridden by a descriptor file. In manual transactions, place the `jdbc:isolation-level` attribute on the first element in the transaction. This attribute is ignored on subsequent elements. In the following example. the attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr" jdbc:transaction-id="0"
                          jdbc:isolation-level="serializable">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement jdbc:transaction-type="manual"
                    jdbc:transaction-id="0">
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
                  WHERE idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
INSERT INTO indirect.usr(lname) VALUES('Doe');
UPDATE indirect.usr SET fname = 'John' WHERE idu = 1;
COMMIT; -- explicit commit
```

Statement Type

The Subscriber channel executes embedded SQL statements, but it doesn't understand them. The JDBC 1 interface defines several methods for executing different types of SQL statements. The following table contains these methods:

Table 12-1 *Methods for Executing SQL Statements*

Statement Type	Method Executed
SELECT	java.sql.Statement.executeQuery(String query):java.sql.ResultSet
INSERT	java.sql.Statement.executeUpdate(String update):int
UPDATE	java.sql.Statement.executeUpdate(String update):int
DELETE	java.sql.Statement.executeUpdate(String update):int
CALL or EXECUTE SELECT INSERT UPDATE DELETE	java.sql.Statement.execute(String sql):boolean

The simplest solution is to map all SQL statements to the `java.sql.Statement.execute(String sql):boolean` method. By default, the Subscriber channel uses this method.

Some third-party drivers, particularly Oracle's JDBC drivers, incorrectly implement the methods used to determine the number of result sets that this method generates. Consequently, the driver can get caught in an infinite loop leading to high CPU utilization. To circumvent this problem, you can use the `jdbc:type` attribute on any `<jdbc:statement>` element to map the SQL statements contained in it to the following methods instead of the default method:

- `java.sql.Statement.executeQuery(String query):java.sql.ResultSet`
- `java.sql.Statement.executeUpdate(String update):int`

The `jdbc:type` attribute has two values: `update` and `query`. For `INSERT`, `UPDATE`, or `DELETE` statements, set the value to `update`. For `SELECT` statements, set the value to `query`. In the absence of this attribute, the driver maps all SQL statements to the default method. If placed on any element other than `<jdbc:statement>`, this attribute is ignored.

Recommendations:

- Place the `jdbc:type="query"` attribute value on all `SELECT` statements.
- Place the `jdbc:type="update"` attribute value on all `INSERT, UPDATE,` and `DELETE` statements.
- Place no attribute value on stored procedure/function calls.

The following XML shows an example of the `jdbc:type` attribute. The attribute is in bold text.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement jdbc:type="update">
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
                  WHERE idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

SQL Queries

To fully support the query capabilities of a database and avoid the difficulty of translating native SQL queries into an XDS format, the driver supports native SQL query processing. You can embed select statements in XDS documents in exactly the same way as any other SQL statement.

For example, assume that the table `usr` has the following contents:

Table 12-2 *Example Contents*

idu	fname	lname
1	John	Doe

The XML document below results in an output document containing a single result set.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <jdbc:statement jdbc:type="query">
        <jdbc:sql>SELECT * FROM indirect.usr</jdbc:sql>
    </jdbc:statement>
</input>

<output xmlns:jdbc="urn:dirxml:jdbc">
    <jdbc:result-set jdbc:number-of-rows="1">
        <jdbc:row jdbc:number="1">
            <jdbc:column jdbc:name="idu"
                         jdbc:position="1"
                         jdbc:type="java.sql.Types.BIGINT
                <jdbc:value>l</jdbc:value>
            </jdbc:column>
            <jdbc:column jdbc:name="fname"
                         jdbc:position="2"
                         jdbc:type="java.sql.Types.VARCHAR>
                <jdbc:value>John</jdbc:value>
            </jdbc:column>
            <jdbc:column jdbc:name="lname"
                         jdbc:position="3"
                         jdbc:type="java.sql.Types.VARCHAR>
                <jdbc:value>Doe</jdbc:value>
            </jdbc:column>
        </jdbc:row>
    </jdbc:result-set>
    <status level="success"/>
</output>
```

SQL queries always produce a single `<jdbc:result-set>` element whether or not the result set contains any rows. If the result set is empty, the `jdbc:number-of-rows` attribute is set to zero.

You can embed more than one query in a document. SQL queries don't require that the referenced tables/views in the synchronization schema be visible to the driver. However, XDS queries do.

If you are building an event to be sent via the Command Processor instead of part of the regular event flow, you need to build the XML in a nodeset variable and use the Parse XML token before issuing the command. For more information, see "XML Parse" in the OpenText™ Identity Manager - Using Designer to Create Policies (https://www.netiq.com/documentation/identity-manager-4.10/policy_designer/data/using-designer-to-create-policies.html).

Example logic XML:

```
<policy xmlns:cmd="http://www.novell.com/nxsl/java/
com.novell.nds.dirxml.driver.XdsCommandProcessor">
  <rule>
    <description>generate SQL Select Statement</description>
    <conditions>
      <and>
        <if-class-name op="equal">User</if-class-name>
      </and>
    </conditions>
    <actions>
      <do-set-local-variable name="sqlstatement">
        <arg-node-set>
          <token-xml-parse>
            <token-text xml:space="preserve">&lt;input
xmlns:jdbc="urn:dirxml:jdbc">&lt;jdbc:statement
jdbc:type="query">&lt;jdbc:sql></token-text>
            <token-text xml:space="preserve">SELECT * FROM lab.users;</
token-text>
            <token-text xml:space="preserve">&lt;/jdbc:sql>&lt;/
jdbc:statement>&lt;/input></token-text>
          </token-xml-parse>
        </arg-node-set>
      </do-set-local-variable>
      <do-trace-message color="yellow" level="1">
        <arg-string>
          <token-xpath expression="cmd:execute($destCommandProcessor,
$sqlstatement)"/>
        </arg-string>
      </do-trace-message>
      <do-veto/>
    </actions>
  </rule>
</policy
```

NOTE: The queries on the Publisher channel with srcCommandProcessor are scheduled for execution on the Subscriber channel and the script processing does not wait for the result to become available.

Data Definition Language (DDL) Statements

Generally, it is not possible to run a Data Definition Language (DDL) statement in a database trigger because most databases do not allow mixed DML and DDL transactions. Although virtual triggers do not overcome this transactional limitation, they do allow DDL statements to be executed as a side effect of an XDS event.

For example:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>CREATE USER jdoe IDENTIFIED BY novell</jdbc:sql>
    </jdbc:statement>
</input>
```

This XML resolves to:

```
SET AUTOCOMMIT OFF
INSERT INTO indirect.usr(fname, lname) VALUES('John', 'Doe');
COMMIT; -- explicit commit
SET AUTOCOMMIT ON
CREATE USER jdoe IDENTIFIED BY novell;
-- implicit commit
```

Using the `jdbc:transaction-id` and `jdbc:transaction-type` attributes to group DML and DDL statements into a single transaction causes the transaction to be rolled back on most databases. Because DDL statements are generally executed as separate transactions, it is possible that the `insert` statement in the above example might succeed and the create user statement might roll back.

It is not possible, however, that the `insert` statement fails and the `create user` statement succeeds. The driver stops executing chained transactions at the point where the first transaction is rolled back.

Logical Operations

Because it is not generally possible to mix DML and DDL statements in a single transaction, a single event can consist of one or more transactions. You can use the `jdbc:op-id` and `jdbc:op-type` to group multiple transactions together into a single logical operation. When grouped in this way, all members of the operation are handled as a single unit with regard to status. If one member has an error, all members return the same status level. Similarly, all members share the same status type.

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr" jdbc:op-id="0"
                          jdbc:op-type="password-set-operation">
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
        <password>Doe{$idu}</password>
    </add>
    <jdbc:statement jdbc:op-id="0">
        <jdbc:sql>CREATE USER jdoe IDENTIFIED BY {$$password}
        </jdbc:sql>
    </jdbc:statement>
</input>
```

The `jdbc:op-type` attribute is ignored on all elements except the first element in a logical operation.

Implementing Password Set with Embedded SQL

Initially setting a password is usually accomplished by creating a database user account. Assuming that an <add> event is generated on the Subscriber channel, the following is an example of the output generated by XSLT style sheets that implement password set as a side effect of an XDS <add> event:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr" jdbc:op-id="0"
                          jdbc:op-type="password-set-operation">
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
        <password>Doe{$idu}</password>
    </add>
    <jdbc:statement jdbc:op-id="0">
        <jdbc:sql>CREATE USER jdoe IDENTIFIED BY {$$password}
        </jdbc:sql>
    </jdbc:statement>
</input>
```

The <add> event is logically bound to the `CREATE USER` DDL statement by the `jdbc:op-id` and `jdbc:op-type` attributes.

The `User DDL` Command Transformation style sheet in the `example.xml` configuration file contains sample XSLT templates that bind user account creation DDL statements to <add> events for all databases that support them.

Implementing Modify Password with Embedded SQL

Modifying a password is usually accomplished by altering an existing database user account. Assuming that a `<modify-password>` event is generated on the Subscriber channel, the following is an example of the output generated by XSLT style sheets that implement modify-password:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <modify-password jdbc:op-id="0"
                     jdbc:op-type="password-set-operation">
       <password>new password</password>
    </modify-password>
    <jdbc:statement jdbc:op-id="0">
        <jdbc:sql>ALTER USER jdoe IDENTIFIED BY {$$password}
        </jdbc:sql>
    </jdbc:statement>
</input>
```

NOTE: Some databases, such as Sybase Adaptive Server Enterprise and Microsoft SQL Server, differentiate between user account names and login account names. Therefore, you might need to supply the login name instead of the username.

The `<modify-password>` event is logically bound to the `ALTER USER` DDL statement by the `jdbc:op-id` and `jdbc:op-type` attributes.

The `User DDL` Command Transformation style sheet in the example `.xml` configuration contains sample XSLT templates that bind password maintenance DDL statements to `<modify-password>` events for all databases that support them.

Implementing Check Object Password

Unlike password set, check object password does not require embedded SQL statements or attributes. Only a user account name is required. This could be obtained from an association value (assuming that associations are being maintained manually), a directory attribute, or a database field. If stored in the directory or database, a query must be issued to retrieve the value.

The example `.xml` configuration file stores database user account names in database fields.

NOTE: Some databases, such as Sybase Adpative Server Enterprise and Microsoft SQL Server, differentiate between user account names and login account names. Therefore, you might need to store two names, not just one.

To implement check object password, append a `dest-dn` attribute value to the `<check-object-password>` event. In the following example, the `dest-dn` attribute is bolded:

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <check-object-password dest-dn="jdoe">
        <password>whatever</password>
    </check-object-password>
</input>
```

Calling Stored Procedures and Functions

The JDBC driver enables you to use stored procedures. The ability to use the `<jdbc:call-procedure>` and `<jdbc:call-function>` elements to call stored procedures from a policy has been tested only against Oracle and is supported only on that platform.

- "Using Embedded SQL to Call Stored Procedures or Functions" on page 156
- "Using the jdbc:call-procedure Element" on page 157
- "Using the jdbc:call-function Element" on page 161

Using Embedded SQL to Call Stored Procedures or Functions

You can call stored procedures or functions in one of two ways:

- Call the procedure or function by using a Statement object.
- Call the procedure by using a Callable Statement object.

Example 1: Calling a Stored Procedure by Using a Statement

```
<!-- call syntax is Oracle -->
<jdbc:statement>
     <jdbc:sql>CALL schema.procedure-name</jdbc:sql/>
</jdbc:statement>
```

Example 2: Calling a Stored Procedure as a CallableStatement

```
<!-- call syntax is vendor agnostic -->
<jdbc:statement>
 <jdbc:call-procedure jdbc:name="schema.procedure-name"/>
</jdbc:statement>
```

Example 3: Calling a Function by Using a Statement

```
<!-- call syntax is Informix -->
<jdbc:statement>
  <jdbc:sql>EXECUTE FUNCTION schema.function-name</jdbc:sql/>

</jdbc:statement>
```

Example 4: Calling a Function as a CallableStatement

```
<!-- call syntax is vendor agnostic -->
<jdbc:statement>
     <jdbc:call-function jdbc:name="schema.function-name"/>
</jdbc:statement>
```

The principle advantage of using the CallableStatement interface is that you do not need to know the proprietary call syntaxes of each database vendor or JDBC implementation. Other advantages include the following:

- It's much easier to build procedure or function calls in the Policy Builder.
- You can differentiate between Null and empty string parameter values.

- You can call functions on all database platforms.

  Oracle, for instance, doesn't support calling functions by using a statement.
- You can retrieve Out parameter values from stored procedure calls.

Using the jdbc:call-procedure Element

Stored procedures do not necessarily require parameters. Only a name is required. If a database supports schemas, we recommend that you schema-qualify the name. If a schema qualifier isn't provided, how names are resolved depends upon your third-party JDBC implementation and might change, depending upon driver configuration settings.

The jdbc:call-procedure element must be wrapped in a jdbc:statement element.

- "Specifying a Procedure Name" on page 157
- "Passing In or In Out Parameter Values" on page 157
- "Handling Out or In Out Parameters" on page 158
- "Example Complex Stored Procedure Calls" on page 159

Specifying a Procedure Name

```
<jdbc:call-procedure jdbc:name="schema.procedure-name"/>
```

Passing In or In Out Parameter Values

The number of jdbc:param elements specified must match the number of param elements declared in the procedure. Only jdbc:param elements corresponding to In or In Out procedure parameters can have values. Out parameters (those that can't be passed values) must be represented by an empty jdbc:param element.

- "Calling a Procedure with No Parameters" on page 157
- "Calling a Procedure with a Null Parameter" on page 157
- "Calling a Procedure with an Empty String Parameter" on page 158
- "Calling a Procedure with a Literal Value" on page 158
- "Calling a Procedure with an Out Parameter" on page 158

Calling a Procedure with No Parameters

```
<jdbc:statement>
     <jdbc:call-procedure jdbc:name="schema.procedure-name"/>
</jdbc:statement>
```

Calling a Procedure with a Null Parameter

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
     <!-- no value element = pass null -->
     <jdbc:param/>
</jdbc:call-procedure>
```

Calling a Procedure with an Empty String Parameter

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
     <!-- empty value element = pass empty string -->
     <jdbc:param>
          <jdbc:value/>
     </jdbc:param>
<jdbc:param>
```

Literals can be passed only to procedure parameters declared as In or In Out. Passed literals must be type-compatible with declared procedure parameters.

Calling a Procedure with a Literal Value

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
     <!-- non-empty value element = pass literal -->
     <jdbc:param>
          <jdbc:value>literal</jdbc:value>
     </jdbc:param>
<jdbc:param>
```

Calling a Procedure with an Out Parameter

Assuming that a procedure has two parameters, the first Out and the second In, you invoke the procedure as follows:

```
<jdbc:call-procedure jdbc:name="schema.procedure-name">
     <!-- the OUT param place -->
     <jdbc:param/>
       <!-- the IN param -->
     <jdbc:param>
          <jdbc:value>literal</jdbc:value>
     </jdbc:param>
<jdbc:param>
```

Handling Out or In Out Parameters

Stored procedures with Out or In Out parameters can return values. These values are returned by the driver and are accessible to policies. Out or In Out parameters values are returned at the same position as their corresponding declared parameter.

Also, to facilitate correlation of procedure calls and output parameter values, Out parameters contain the same event-ID value as the procedure call that generated them. This is particularly useful when multiple calls are made in the same document.

- “Null or No Return Value” on page 158
- “Empty String Return Value” on page 159
- “Literal Return Value” on page 159

Null or No Return Value

Assuming that a procedure has a single Out or In parameter, the following output is generated:

```
<output>
    <!-- no value element = OUT param returned null or IN param -->
   <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
        <jdbc:param/>
   </jdbc:out-parameters>
   <status event-id="0" level="success"/>
</output>
```

Empty String Return Value

Assuming that a procedure has a single Out or In Out parameter, the following output is generated:

```
<output>
    <!-- empty value element = returned empty string -->
    <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
        <jdbc:param>
            <jdbc:value/>
        </jdbc:param>
     </jdbc:out-parameters>
     <status event-id="0" level="success"/>
</output>
```

Literal Return Value

Assuming that a procedure has a single Out or In Out parameter, the following output is generated:

```
<output>
    <!-- no-empty value element = returned literal value -->
    <jdbc:out-parameters event-id="0" jdbc:number-of-params="2">
        <jdbc:param>
            <jdbc:value>literal<jdbc:value>
        </jdbc:param>
    </jdbc:out-parameters>
    <status event-id="0" level="success"/>
</output>
```

Example Complex Stored Procedure Calls

- "Procedure Declaration" on page 159
- "Procedure Call from Policy" on page 160
- "Procedure Output to Policy" on page 160

Procedure Declaration

This procedure uses Oracle PSQL syntax.

```
CREATE PROCEDURE indirect.p1(i1 IN VARCHAR2, io2 IN OUT VARCHAR2, o3 OUT
INTEGER, i4 IN VARCHAR2)
AS
BEGIN
    SELECT 'literal' INTO io2 FROM DUAL;
    SELECT 1 INTO o3 FROM DUAL;
END p1;
```

Procedure Call from Policy

```
<input>
     <jdbc:statement event-id="0">
         <jdbc:call-procedure jdbc:name="indirect.p1">
              <!-- i1 IN VARCHAR2 -->
              <jdbc:param>
                   <!-- pass empty string -->
                   <jdbc:value/>
              </jdbc:param>
              !-- io2 IN OUT VARCHAR2 -->
              <jdbc:param>
                   <!-- pass literal -->
                   <jdbc:value>literal</jdbc:value>
              </jdbc:param>
              <!-- o3 OUT INTEGER -->
              <!-- param placeholder -->
              <jdbc:param/>
              <!-- o4 IN VARCHAR2 -->
              <!-- pass null -->
              <jdbc:param/>
          </jdbc:call-procedure>
     </jdbc:statement>
</input>
```

Procedure Output to Policy

```
<output>
     <jdbc:out-parameters event-id="0" jdbc:number-of-params="2">
          <jdbc:param/>
          <jdbc:param jdbc:name="IO2"
                    jdbc:param-type="INOUT"
                    jdbc:position="2"
                    jdbc:sql-type="java.sql.Types.VARCHAR">
               <jdbc:value>literal</jdbc:value>
          </jdbc:param>
          <jdbc:param jdbc:name="O3"
                    jdbc:param-type="OUT"
                    jdbc:position="3"
                    jdbc:sql-type="java.sql.Types.DECIMAL">
                <jdbc:value>1</jdbc:value>
          </jdbc:param>
          <jdbc:param/>
     </jdbc:out-parameters>
     <status event-id="0" level="success"/>
</output>
```

Using the jdbc:call-function Element

Functions do not necessarily require parameters. Only a name is required. If a database supports schemas, we recommend that you schema-qualify the name. If a schema qualifier isn't provided, how names are resolved depends upon your third-party JDBC implementation and might change depending upon driver configuration settings.

The jdbc:call-function element must be wrapped in a jdbc:statement element.

- "Specifying a Function Name" on page 161
- "Passing In Parameter Values" on page 161
- "Calling a Function with No Parameter" on page 161
- "Calling a Function with a Null Parameter" on page 161
- "Calling a Function with an Empty String Parameter" on page 161
- "Calling a Function with a Literal Value" on page 162
- "Handling Function Results" on page 162
- "Example Complex Function Calls" on page 164

Specifying a Function Name

```
<jdbc:call-function jdbc:name="schema.function-name"/>
```

Passing In Parameter Values

The number of jdbc:param elements specified must match the number of params declared in the function.

Calling a Function with No Parameter

```
<jdbc:call-function jdbc:name="schema.function-name"/>
```

Calling a Function with a Null Parameter

```
<jdbc:call-function jdbc:name="schema.function-name">
     <!-- no value element = null -->
     <jdbc:param/>
</jdbc:call-procedure>
```

Calling a Function with an Empty String Parameter

```
<jdbc:call-function jdbc:name="schema.function-name">
     <!-- empty value element = pass empty string -->
     <jdbc:param>
          <jdbc:value/>
     </jdbc:param>
<jdbc:param>
```

Literals can be passed to function parameters declared as In. Passed literals must be type-compatible with declared function parameters.

Calling a Function with a Literal Value

```
<jdbc:call-function jdbc:name="schema.function-name">
     <!-- non-empty value element = pass literal -->
     <jdbc:param>
          <jdbc:value>literal</jdbc:value>
     </jdbc:param>
<jdbc:param>
```

Handling Function Results

Unlike stored procedures, functions do not support Out or In Out parameters. They can, however, return a single, scalar value (such as an integer or string) or return a result set. Also, to facilitate correlation of function calls and results, results contain the same event-id value as the function call that generated them. This is particularly useful when multiple calls are made in the same document.

- "Scalar Return Value" on page 162
- "Empty Set" on page 162
- "Non-Empty Results Set" on page 163
- "Multiple Result Sets" on page 163
- "Oracle Results Set" on page 163
- "Returning Result Sets as Out Parameters" on page 163
- "Special Attributes" on page 163

Scalar Return Value

Scalar return values are returned by using the same syntax as stored procedure Out parameters. The scalar return value is always returned in the first parameter position.

```
<output>
     <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
          <jdbc:param jdbc:name="return value"

                    jdbc:param-type="OUT"
                    jdbc:position="1"
                    jdbc:sql-type="java.sql.Types.VARCHAR">
               <jdbc:value>1</jdbc:value>
          </jdbc:param>
     </jdbc:out-parameters>
     <status event-id="0" level="success"/>
</output
```

Empty Set

Assuming that a function returns no results set or an empty result set, the following output is generated:

```
<output>
     <jdbc:result-set event-id="0" jdbc:number-of-rows="0"/>
     <status event-id="0" level="success"/>
</output>
```

Non-Empty Results Set

Assuming a function returns a non-empty result set, output similar to the following is generated:

```
<output>
     <jdbc:result-set event-id="0" jdbc:number-of-rows="1">
          <jdbc:row jdbc:number="1">
               <jdbc:column jdbc:name="SYSDATE"
                         jdbc:position="1
                         jdbc:type="java.sql.Types.TIMESTAMP">
                    <jdbc:value>2007-01-03 14:52:20.0</jdbc:value>
               </jdbc:column>
          </jdbc:row>
     </jdbc:result-set>
     <status event-id="0" level="success"/>
</output>
```

Multiple Result Sets

Multiple result sets are returned in the order returned by the function. They all share a common event-id value.

```
<output>
     <jdbc:result-set event-id="0" jdbc:number-of-rows="0"/>
     <jdbc:result-set event-id="0" jdbc:number-of-rows="0"/>
     <status event-id="0" level="success"/>
</output>
```

Oracle Results Set

Oracle's JDBC implementation uses a proprietary mechanism to return a single result set from a function. To return a result set from an Oracle function, you need to explicitly set the jdbc:return-type value to OracleTypes.CURSOR on the jdbc:call-function element.

Returning Result Sets as Out Parameters

See the special attribute “jdbc:return-format” on page 163.

Special Attributes

jdbc:return-format

This attribute can be placed on the jdbc:call-function element to format the first row of a returned results set as stored procedure Out parameters of the result.

This works only when the jdbc:return-type attribute isn't used.

```
<input>
     <jdbc:statement>
          <jdbc:call-function jdbc:name="schema.function-name"
                    jdbc:return-format="return value">
               </jdbc:call-function>
     </jdbc:statement>
</input>
```

jdbc:return-type

This attribute can be placed on the jdbc:call-function element to allow Oracle functions to return a result set.

```
<input>
     <jdbc:statement>
          <jdbc:call-function jdbc:name="schema.function"
               jdbc:return-type="OracleTypes.CURSOR">
          </jdbc:call-function>
     </jdbc:statement>
</input>
```

Example Complex Function Calls

- "Function Declaration" on page 164
- "Function Call from a Policy" on page 164
- "Function Results to a Policy" on page 165

Function Declaration

This declaration is for Oracle PSQL syntax.

```
CREATE OR REPLACE FUNCTION indirect.f1(i1 IN VARCHAR2, i2 IN INTEGER)
  RETURN VARCHAR2
AS
  o_idu VARCHAR2(32);
BEGIN
  SELECT 'literal' INTO o_idu FROM DUAL;
  RETURN o_idu;
END f1;
```

Function Call from a Policy

```
<input>
     <jdbc:statement>

          <jdbc:call-function jdbc:name="indirect.f1">
               <jdbc:param>
                    <jdbc:value>literal</jdbc:value>
               </jdbc:param>
               <jdbc:param>
                    <jdbc:value>1</jdbc:value>
               </jdbc:param>
           </jdbc:call-function>
     </jdbc:statement>
</input>
```

Function Results to a Policy

```
<output>
     <jdbc:out-parameters event-id="0" jdbc:number-of-params="1">
          <jdbc:param jdbc:name="return value"
                    jdbc:param-type="OUT"
                    jdbc:position="1"
                    jdbc:sql-type="java.sql.Types.VARCHAR">
               <jdbc:value>literal</jdbc:value>
          </jdbc:param>
     </jdbc:out-parameters>
     <status event-id="0" level="success"/>
</output>
```

Best Practices

For performance reasons, it is better to call a single stored procedure/function that contains multiple SQL statements than to embed multiple statements in an XDS document.

In the following examples, the single stored procedure or function is preferred.

Single Stored Procedure

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="fname">
            <value>John</value>
        </add-attr>
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>CALL PROCEDURE set_name('John', 'Doe')</jdbc:sql>
    </jdbc:statement>
</input>
```

Multiple Embedded Statements

```
<input xmlns:jdbc="urn:dirxml:jdbc">
    <add class-name="usr">
        <add-attr name="lname">
            <value>Doe</value>
        </add-attr>
    </add>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET fname = 'John'
WHERE idu = {$idu}</jdbc:sql>
    </jdbc:statement>
    <jdbc:statement>
        <jdbc:sql>UPDATE indirect.usr SET lname = 'Doe'
WHERE idu = {$idu}</jdbc:sql>
    </jdbc:statement>
</input>
```

The syntax used to call stored procedures or functions varies by database. For additional information, see "Syntaxes for Calling Stored Procedures and Functions" on page 170.

13 Supported Databases

- "Database Interoperability" on page 167
- "Supported Databases" on page 167
- "Database Characteristics" on page 168

Database Interoperability

The Identity Manager Driver for JDBC is designed to inter-operate with a specific set of JDBC driver implementations, instead of a specific set of databases. Consequently, the list of supported databases is primarily driven by the capabilities of supported third-party JDBC drivers. A secondary factor is testing resources.

Supported Databases

The following databases or database versions have been tested and are recommended for use with this product:

Table 13-1 *Supported Databases*

Database	Minor Version
IBM DB2 Universal Database (UDB)	9.0 or later
Informix Dynamic Server (IDS)	11.0 or later
Microsoft SQL Server	2008, 2008 R2, 2012, 2014, 2016, 2017, and 2019
MySQL	5.5.x or later
	5.6.x or later
	8.x
Oracle	12c, 18c and 19c
Azure Database for PostgrSQL flexible servers	14.4
PostgreSQL	14.0 or later
	12.2 or later
	12.8 or later
	11.7 or later
Sybase Adaptive Server Enterprise (ASE)	15.0 or later
MariaDB	10.2.13

You can use the JDBC driver with other databases or database versions. However, NetIQ does not support them. To interoperate with the JDBC driver, a database must meet the following requirements:

- Support the SQL-92 entry level grammar.
- Be JDBC-accessible.

Database Characteristics

- "Database Features" on page 168
- "Current Time Stamp Statements" on page 169
- "Syntaxes for Calling Stored Procedures and Functions" on page 170
- "Left Outer Join Operators" on page 171
- "Undelimited Identifier Case Sensitivity" on page 171
- "Supported Transaction Isolation Levels" on page 172
- "Commit Keywords" on page 172
- "IBM DB2 Universal Database (UDB)" on page 173
- "Informix Dynamic Server (IDS)" on page 174
- "Microsoft SQL Server" on page 175
- "MySQL/MariaDB" on page 175
- "Oracle" on page 176
- "PostgreSQL" on page 177
- "Sybase Adaptive Server Enterprise (ASE)" on page 178

Database Features

Table 13-2 *Database Features*

Database	Schemas	Views	Identity Columns	Sequences	Stored Procedures	Functions	Triggers	Instead-Of-Triggers
IBM DB2 UDB 9.x	Y	Y	Y	N	Y[1]	Y[1]	Y	Y
Informix IDS 11.x	Y	Y	Y[2]	N	Y[3]	Y	Y	Y
MS SQL 2008, 2008 R2, 2012, 2014, and 2016	Y	Y	Y	N	Y	Y	Y	Y

Database	Schemas	Views	Identity Columns	Sequences	Stored Procedures	Functions	Triggers	Instead-Of-Triggers
MySQL 5.5.x and 5.6.x	Y	Y	Y[4]	N	Y	Y	Y	N
Oracle 12c, 18c and 19c	Y	Y	N	Y	Y	Y	Y	Y
Postgres 8.4.x, 9.0.x, and 9.6.3	Y	Y	Y[5]	Y	Y	Y	Y[6]	Y[6]
Sybase ASE 15.0	Y	Y	Y	N	Y	N	Y	N
MariaDB 10.2.13	Y	Y	Y[4]	N	Y	Y	Y	N

[1] DB2 natively supports stored procedures or functions written in Java. To write procedures by using the native SQL procedural language, install a C compiler on the database server.

[2] The Informix identity column keyword is `SERIAL8`.

[3] Informix stored procedures cannot return values through OUT parameters.

[4] The identity column keyword is `AUTO_INCREMENT` for MySQL and MariaDB.

[5] You can use a PostgreSQL sequence object to provide default values for primary key columns, effectively simulating an identity column.

[6]PostgreSQL has a native construct called rules. This construct can be used to effectively simulate triggers and instead-of-triggers. It also supports the use of triggers or instead-of-triggers written in a variety of procedural programming languages.

Current Time Stamp Statements

The following table lists SQL statements used to retrieve the current date and time by database:

Table 13-3 Time Stamp Statements

Database	Current Time Stamp Statement	ANSI-Compliant
IBM DB2 UDB	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY	No
Informix IDS	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES	No
MSSQL	SELECT (CURRENT_TIMESTAMP)	Yes
MySQL/ MariaDB	SELECT (CURRENT_TIMESTAMP)	Yes
Oracle	SELECT (SYSDATE) FROM SYS.DUAL	No
PostgreSQL	SELECT (CURRENT_TIMESTAMP)	Yes
Sybase ASE	SELECT GETDATE()	No

Syntaxes for Calling Stored Procedures and Functions

The following table lists the syntaxes for calling a stored procedure or function by database vendor. There's also a vendor-neutral JDBC escape syntax (see JDBC Escape Syntax (http://edocs.bea.com/wls/docs81/jdbc_drivers/sqlescape.html)). Whenever possible, it is more secure to call a stored procedure or function by using the jdbc:call-function or jdbc:call-procedure syntax. See "Calling Stored Procedures and Functions" on page 156.) Other syntaxes should be used only when specifying procedure or function calls in driver parameters (for example, "Post Polling Statements" on page 98 and "Connection Initialization Statements" on page 69).

Table 13-4 Calling a Stored Procedure or Function

Database	Stored Procedure/Function JDBC Call Syntax
IBM DB2 UDB	{call *schema-name.procedure-name*(*parameter-list*)}
Informix IDS	EXECUTE [PROCEDURE | FUNCTION] *schema-name.routine-name*(*parameter-list*)
MSSQL	EXECUTE *schema-name.procedure-name*(*parameter-list*)
MySQL/ MariaDB	CALL *schema-name.procedure-name*(*parameter-list*)
Oracle[1]	CALL *schema-name.procedure-name*(*parameter-list*)
PostgreSQL	SELECT *schema-name.procedure-name*(*parameter-list*)
Sybase ASE	EXECUTE *schema-name.procedure-name*(*parameter-list*)

[1] Oracle's JDBC implementation does not support calling functions as a string.

Left Outer Join Operators

The following table lists outer join operators by database.

Table 13-5 *Outer Join Operators*

Database	Left Outer Join Operator	ANSI-Compliant
IBM DB2 UDB	LEFT OUTER JOIN	Yes
Informix IDS	LEFT OUTER JOIN	Yes
MSSQL 2005	LEFT OUTER JOIN	Yes
MySQL/MariaDB	LEFT OUTER JOIN	Yes
Oracle	LEFT OUTER JOIN	Yes
PostgreSQL	LEFT OUTER JOIN	Yes
Sybase ASE	*=	No

Undelimited Identifier Case Sensitivity

Table 13-6 *Case Sensitivity for Undelimited Identifiers*

Database	Case-Sensitive?
IBM DB2 UDB	No
Informix IDS	No
MSSQL	No
MySQL/ MariaDB	Yes
Oracle	No
PostgreSQL	No
Sybase ASE	Yes

Supported Transaction Isolation Levels

Table 13-7 *Supported Transaction Isolation Levels*

Database	None	Read Uncommitted	Read Committed	Repeatable Read	Serializable	URL
IBM DB2 UDB	N	Y	Y[1]	Y	Y	Setting JDBC Transaction Isolation Levels (http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/tjvjdiso.htm)
MySQL/ MariaDB (InnoDB* Table Type)	N	Y	Y	Y[1]	Y	InnoDB Transaction Isolation Levels (http://dev.mysql.com/doc/mysql/en/innodb-transaction-isolation.html)
Oracle	N	N	Y[1]	N	Y	JDBC Transaction Optimization (http://www.oracle.com/technology/oramag/oracle/02-jul/o42special_jdbc.html)
PostgreSQL	N	N[2]	Y[1]	N[2]	Y	Transaction Isolation (http://www.postgresql.org/docs/current/static/transaction-iso.html)

[1] This is the default isolation level for this database. [2] Can be set, but it is aliased to a supported isolation level.

Commit Keywords

The following table identifies the commit keywords for supported databases:

Table 13-8 Commit Keywords

Database	Commit Keyword
IBM DB2 UDB	COMMIT
Informix IDS	COMMIT WORK[1]
MSSQL	GO
MySQL/ MariaDB	COMMIT
Oracle	COMMIT
PostgreSQL	COMMIT
Sybase ASE	GO

[1] For logging and ANSI-compliant databases. Non-logging databases do not support transactions.

IBM DB2 Universal Database (UDB)

- "Database Properties" on page 173
- "Dynamic Defaults" on page 173
- "Known Issues" on page 174

Database Properties

Table 13-9 Properties for IBM DB2 UDB

Property	Value
Current Timestamp Statement	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Dynamic Defaults

The following table lists database compatibility parameters that the JDBC driver implicitly sets at runtime. Do not explicitly override these settings.

Table 13-10 Dynamically Configured IBM DB2 Universal Database Settings

Display Name	Tag Name	Value
Current Timestamp Statement:	current-timestamp-stmt	SELECT (CURRENT TIMESTAMP) FROM SYSIBM.SYSDUMMY1 FETCH FIRST 1 ROW ONLY
Timestamp Translator class:	time-translator-class	com.novell.nds.dirxml.driver.jdbc.db.DB2Timestamp

Known Issues

The timestamp format is proprietary. See "Known Issues" on page 179.

Informix Dynamic Server (IDS)

- "Database Properties" on page 174
- "Dynamic Defaults" on page 174
- "Known Issues" on page 175

Database Properties

Table 13-11 Settings for Informix Dynamic Server

Property	Value
Current Timestamp Statement	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES
Case-Sensitive?	No
Commit Keyword	COMMIT WORK[1]
Left Outer Join Operator	LEFT OUTER JOIN

[1] For logging and ANSI-compliant databases. Nonlogging databases do not support transactions.

Dynamic Defaults

The following table lists database compatibility parameters that the JDBC driver implicitly sets at runtime. Do not explicitly overwrite these settings.

Table 13-12 Dynamically Configured Informix Dynamic Server Settings

Display Name	Tag Name	Value
Current Timestamp Statement:	current-timestamp-stmt	SELECT FIRST 1 (CURRENT YEAR TO FRACTION(5)) FROM INFORMIX.SYSTABLES

Known Issues

- `NUMERIC` or `DECIMAL` columns cannot be used as primary keys unless the scale (the number of digits to the right of the decimal point) is explicitly set to 0 when the table is created. By default, the scale is set to 255.
- DBAs cannot grant privileges to objects they don't own.

Microsoft SQL Server

- "Database Properties" on page 175
- "Dynamic Defaults" on page 175

Database Properties

Table 13-13 *Settings for Microsoft SQL Server*

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Case-Sensitive?	No
Commit Keyword	GO
Left Outer Join Operator (2005,2008,2008 R2, 2012, 2014)	LEFT OUTER JOIN

Dynamic Defaults

The following table lists database compatibility parameters that the JDBC driver implicitly sets at runtime. Do not explicitly overwrite these settings.

Table 13-14 *Dynamically Configured Microsoft SQL Server Settings*

Display Name	Tag Name	Value
Add default values on insert?	add-default-values-on-view-insert	True
Left outer-join operator (2005,2008,2008 R2, 2012, 2014)	left-outer-join-operator	LEFT OUTER JOIN

MySQL/MariaDB

- "Database Properties" on page 176
- "Dynamic Defaults" on page 176
- "Known Issues" on page 176

Database Properties

Table 13-15 Settings for MySQL/MariaDB

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Case-Sensitive?	Yes
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Dynamic Defaults

The following table lists database compatibility parameters that are dynamically configured at runtime for this database.

Table 13-16 Dynamically Configured MySQL/MariaDB Settings

Display Name	Tag Name	Value
Supports schemas in metadata retrieval?	supports-schemas-in-metadata-retrieval	false

Known Issues

- `TIMESTAMP` columns, when they are updated after being initially set to 0 or `NULL`, are always set to the current date and time. To compensate for this behavior, we recommend that you map Identity Vault Time and Timestamp syntaxes to `DATETIME` columns.

Oracle

- "Database Properties" on page 177
- "Dynamic Defaults" on page 177
- "Limitations" on page 177

Database Properties

Table 13-17 Settings for Oracle

Property	Value
Current Timestamp Statement	SELECT (SYSDATE) FROM SYS.DUAL
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	(+)

Dynamic Defaults

The following table lists database compatibility parameters that the JDBC driver implicitly sets at runtime. Do not explicitly overwrite these settings.

Table 13-18 Dynamically Configured Oracle Settings

Display Name	Tag Name	Value
Left outer-join operator	left-outer-join-operator	(+)
Exclude filter expression	exclude-table-filter	BIN\$.{22}==\$0
Lock statement generator class	lock-generator-class	com.novell.nds.dirxml.driver.jdbc.db.lock.OraLockGenerator

The default exclusion filter omits dropped tables from the synchronization schema.

Limitations

`LONG`, `LONG RAW`, and `BLOB` columns cannot be referenced in a trigger. You can't reference columns of these types by using the `:NEW` qualifier in a trigger, including instead-of-triggers.

PostgreSQL

- "Database Properties" on page 178
- "Known Issues" on page 178

Database Properties

Table 13-19 Settings for PostgreSQL

Property	Value
Current Timestamp Statement	SELECT (CURRENT_TIMESTAMP)
Case-Sensitive?	No
Commit Keyword	COMMIT
Left Outer Join Operator	LEFT OUTER JOIN

Known Issues

PostgreSQL does not support `<check-object-password>` events. You control authentication by manually inserting entries into the `pg_hba.conf` file.

Sybase Adaptive Server Enterprise (ASE)

- "Database Properties" on page 178
- "Dynamic Defaults" on page 178
- "Known Issues" on page 179

Database Properties

Table 13-20 Settings for Sybase ASE

Property	Value
Current Timestamp Statement	SELECT GETDATE()
Case-Sensitive?	Yes
Commit Keyword	GO
Left Outer Join Operator	*=

Dynamic Defaults

The following table lists database compatibility parameters that the JDBC driver implicitly sets at runtime. Do not explicitly overwrite these settings.

Table 13-21 Dynamically Configured Sybase ASE Settings

Display Name	Tag Name	Value
Current timestamp statement	current-timestamp-stmt	SELECT GETDATE()
Left outer-join operator	left-outer-join-operator	*=
Timestamp Translator class	time-translator-class	com.novell.nds.dirxml.driver.jdbc.db.SybaseTimestamp

Known Issues

- Padding and truncation of binary values.

  To ensure ANSI-compliant padding and truncation behavior for binary values, make sure that binary column types (other than `IMAGE`) meet the following criteria:

  - They are exactly the size of the eDirectory™ attribute that maps to them.
  - They are constrained `NOT NULL`.
  - They are added to the Publisher and Subscriber Creation policies.

  If they are constrained `NULL`, trailing zeros, which are significant to eDirectory, are truncated. If binary columns exceed the size of their respective eDirectory attributes, extra 0s are appended to the value.

  The recommended solution is to use only the `IMAGE` data type when synchronizing binary values.
- `DATETIME` fractions of a second are rounded. Sybase Timestamps are at best accurate to 1/300th of a second (approximately.003 seconds). The database server rounds to the nearest 1/300th of a second as opposed to the nearest 1/1000th of a second (.001 seconds or 1 millisecond).
- Timestamp formats are proprietary.

14 Third-Party JDBC Drivers

- "Third-Party JDBC Driver Interoperability" on page 181
- "Third-Party JDBC Driver Types" on page 181
- "Supported Third-Party JDBC Drivers (Recommended)" on page 182
- "Supported Third-Party JDBC Drivers (Not Recommended)" on page 198
- "Deprecated Third-Party JDBC Drivers" on page 202
- "Other Third-Party JDBC Drivers" on page 203
- "Security Issues" on page 204

Third-Party JDBC Driver Interoperability

The Identity Manager JDBC driver is designed to interoperate with a specific set of third-party JDBC drivers, instead of a specific set of databases. In fact, the third-party JDBC driver, not the database, is the primary determinant of whether the JDBC driver works against any given database. As a general rule, if the JDBC driver interoperates well with a given third-party JDBC driver, it interoperates well with databases and database versions that the third-party driver supports.

We strongly recommend that you use the third-party JDBC drivers supplied by major enterprise database vendors whenever possible, such as those listed in "Supported Third-Party JDBC Drivers (Recommended)" on page 182. They are usually free, mature, and known to interoperate well with the JDBC driver and the databases they target. You can use other third-party drivers, but OpenText does not support them.

In general, most third-party drivers are backward compatible. However, even if they are backward compatible, they are generally not forward compatible. Anytime a database server is upgraded, the third-party driver used with this product should probably be updated as well.

Also, as a general rule, we recommend that you use the latest version of a third-party driver, unless otherwise noted.

Third-Party JDBC Driver Types

The following sections describe four third-party JDBC driver types and explains the recommended type for use with the Identity Manager JDBC driver:

- "Driver Types" on page 182
- "Which Type To Use?" on page 182

Driver Types

There are four types of third-party drivers:

- **Type 1:** A third-party JDBC driver that is partially Java and communicates indirectly with a database server through a native ODBC driver.

  Type 1 drivers serve as a JDBC-ODBC bridge. Sun provides a JDBC-ODBC bridge driver for experimental use and for situations when no other type of third-party JDBC driver is available.
- **Type 2:** A third-party JDBC driver that is part Java and communicates indirectly with a database server through its native client APIs.
- **Type 3:** A third-party JDBC driver that is pure Java and communicates indirectly with a database server through a middleware server.
- **Type 4:** A third-party JDBC driver that is pure Java and communicates directly with a database server.

Which Type To Use?

Type 3 and 4 drivers are generally more stable than type 1 and 2 drivers. Type 1 and 2 drivers are generally faster than type 3 and 4 drivers. Type 2 and 3 drivers are generally more secure than type 1 and 4 drivers.

Because Identity Manager uses a directory as its datastore, and because databases are usually significantly faster than directories, performance isn't a primary concern. Stability, however, is an issue. For this reason, we recommend that you use a type 3 or 4 third-party JDBC driver whenever possible.

Because third-party code is being executed within the environment, it is recommended to always use the Remote Loader to execute third-party code on its own to ensure the integrity of the directory process. This also makes upgrades of the shim or third-party code simple and safe because the directory does not need to be restarted.

Supported Third-Party JDBC Drivers (Recommended)

This section discusses supported third-party drivers. Using one of these supported drivers is recommended.

- "Third-Party JDBC Driver Features" on page 183
- "JDBC URL Syntaxes" on page 183
- "JDBC Driver Class Names" on page 184
- "Supported Third-Party Jar File Placement" on page 185
- "IBM DB2 Universal Database Type 4 JDBC Driver" on page 185
- "Informix JDBC Driver" on page 186
- "Microsoft JDBC Driver for SQL Server" on page 188
- "jTDS JDBC Driver" on page 189
- "MySQL Connector/J JDBC Driver" on page 191
- "Oracle Thin Client JDBC Driver" on page 192

- "Oracle OCI JDBC Driver" on page 194
- "PostgreSQL JDBC Driver" on page 195
- "Sybase Adaptive Server Enterprise JConnect JDBC Driver" on page 196
- "MariaDB Connector/J JDBC Driver" on page 197

Additional drivers are supported but not recommended. For a list of these drivers, see "Supported Third-Party JDBC Drivers (Not Recommended)" on page 198.

Third-Party JDBC Driver Features

The following table summarizes third-party JDBC driver features:

Table 14-1 *Third-Party JDBC Driver Features*

Driver	Supports Encrypted Transport?	Supports Retrieval of Auto-Generated Keys?
IBM DB2 UDB Type 4	No	No
Informix	No	No
MySQL Connector/J	Yes	Yes
jTDS	Yes	Yes
Oracle Thin Client	Yes	No
Oracle OCI	Yes	No
PostgreSQL	Yes, for JDBC 3 (Java 1.4) versions and later	No
Sybase jConnect	Yes	No

JDBC URL Syntaxes

The following table lists URL syntaxes for supported third-party JDBC drivers:

Table 14-2 URL Syntaxes

Third-Party JDBC Driver	JDBC URL Syntax
IBM DB2 UDB Type 4, Universal	jdbc:db2:*//ip-address:50000/database-name*
Informix	jdbc:informix-sqli:*//ip-address*:1526/*database-name*:informixserver=*server-id*
jTDS	*jdbc:jtds:sqlserver://ip-address/*database-name
MySQL Connector/J	jdbc:mysql:*//ip-address*:3306/*database-name*
Oracle OCI	jdbc:oracle:oci8:@*tns-name*
Oracle Thin Client	sid - jdbc:oracle:thin:@*ip-address*:1521:<*sid*>
	cdb - jdbc:oracle:thin:@*ip-address*:1521/<service>
PostgreSQL	jdbc:postgresql:*//ip-address*:5432/*database-name*
Sybase jConnect	jdbc:sybase:Tds:*ip-address*:2048/*database-name*

This information is used in conjunction with the Authentication Context parameter. For information on this parameter, see "Authentication Context" on page 57.

JDBC Driver Class Names

The following table lists the fully qualified Java class names of supported third-party JDBC drivers:

Table 14-3 Class Names of Third-Party JDBC Drivers

Third-Party JDBC Driver	Class Name
IBM DB2 UDB Type 4, Universal	com.ibm.db2.jcc.DB2Driver
Informix	com.informix.jdbc.IfxDriver
jTDS	net.sourceforge.jtds.jdbc.Driver
MySQL Connector/J	org.gjt.mm.mysql.Driver
Oracle OCI	oracle.jdbc.driver.OracleDriver
Oracle Thin Client	oracle.jdbc.driver.OracleDriver
PostgreSQL	org.postgresql.Driver
Sybase jConnect 6.0	com.sybase.jdbc3.jdbc.SybDriver
Sybase jConnect 7.0	com.sybase.jdbc4.jdbc.SybDriver

This information is used in conjunction with the JDBC Driver Class Name parameter. For information on this parameter, see "Third-Party JDBC Driver Class Name" on page 60.

Supported Third-Party Jar File Placement

The following tables identify the paths where third-party JDBC driver jar files should be placed on an Identity Manager or Remote Loader server, assuming default installation paths. Ensure to restart eDirectory, after placing the jar files in the corresponding directory, for the change to take effect.

Table 14-4 *Locations for `jar` Files: Identity Manager Server*

Platform	Directory Path
Solaris, Linux, or AIX	`/opt/novell/eDirectory/lib/dirxml/classes` (eDirectory 8.8.x)
Windows	`novell\NDS\lib`

Table 14-5 *Locations for `jar` Files: Remote Loader Server*

Platform	Directory Path
Solaris, Linux, or AIX	`/opt/novell/eDirectory/lib/dirxml/classes` (eDirectory 8.8.x)
Windows	`novell\RemoteLoader\lib`

IBM DB2 Universal Database Type 4 JDBC Driver

- "Driver Information" on page 185
- "Compatibility" on page 186
- "Security" on page 186
- "Known Issues" on page 186

Driver Information

Table 14-6 *IBM DB2 Driver: Type 4*

Supported Database Versions	IBM DB2 9.7 or later
Class Name	com.ibm.db2.jcc.DB2Driver
Type	4
URL Syntax	jdbc:db2:*//ip-address*:50000/*database-name*
Download Instructions	Download as part of the latest FixPack (recommended). IBM Support & Downloads (http://www.ibm.com/support/us/) or Copy the file from the database server. file:///*database-installation-directory*/java

Filename	`db2jcc.jar`, `db2jcc_license_cu.jar`, `db2jcc_javax.jar` (optional), `db2jcc_license_cu-9.5.jar`, `db2jcc-9.5.jar`
Documentation URLs	DB2 Information Center (http://publib.boulder.ibm.com/infocenter/db2help) DB2 Universal JDBC Driver (http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/t0010264.htm) Security under the DB2 Universal JDBC Driver (http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.udb.doc/ad/cjvjcsec.htm)

Unlike the type 3 driver, the type 4 driver has only a minimal set of defined error codes. This absence inhibits the JDBC driver's ability to distinguish between connectivity, retry, authentication, and fatal error conditions.

Compatibility

The IBM DB2 driver is backward compatible. Database server updates are frequent. Driver updates are infrequent.

Security

The IBM DB2 driver supports a variety of authentication security mechanisms but does not support encrypted transport.

Known Issues

It's very difficult to diagnose and remedy Java-related errors on the database server.

Numerous error conditions and error codes can arise when you attempt to install and execute user-defined stored procedures and functions written in Java. Diagnosing these can be time consuming and frustrating. A log file (`db2diag.log` on the database server) can often provide additional debugging information. In addition, all error codes are documented and available online.

Informix JDBC Driver

- "Driver Information" on page 187
- "Compatibility" on page 187
- "Security" on page 187
- "Required Parameter Settings for ANSI-Compliant Databases" on page 187
- "Dynamic Parameter Defaults" on page 188
- "Known Issues" on page 188

Driver Information

Table 14-7 *Informix JDBC Driver*

Supported Database Versions	Dynamic Server 11.5 or later
Class Name	com.informix.jdbc.IfxDriver
Type	4
URL Syntax	jdbc:informix-sqli://*ip-address*:1526/*database-name*:informixserver=*server-id*
Download Instructions	Download URL (http://www-306.ibm.com/software/data/informix/tools/jdbc)
Filenames (11)	`ifxjdbc.jar`, `jdbcx.jar` (optional)
Documentation URLs	Informix Information Center (http://publib.boulder.ibm.com/infocenter/ids9help/index.jsp)
	For more information on Informix JDBC Driver, see Informix Library (http://www-306.ibm.com/software/data/informix/pubs/library/)

Compatibility

The Informix driver is backward compatible. Database server updates and driver updates are infrequent.

Security

The Informix driver does not support encrypted transport.

Required Parameter Settings for ANSI-Compliant Databases

The following table lists driver parameters that must be explicitly set for the JDBC driver to interoperate with the Informix driver against ANSI-compliant databases.

Table 14-8 *Driver Settings for ANSI-Compliant Databases*

Display Name	Tag Name	Value
Supports schemas in metadata retrieval?	supports-schemas-in-metadata-retrieval See "Supports Schemas in Metadata Retrieval?" on page 80.	false
Force username case:	force-username-case See "Force Username Case" on page 78.	upper

Dynamic Parameter Defaults

The following table lists driver compatibility parameters that the JDBC driver implicitly sets at runtime. Do not override these settings.

Table 14-9 *Informix JDBC Settings Not to Override*

Display Name	Tag Name	Value
Function return method:	function-return-method See "Function Return Method" on page 79.	result set

Known Issues

- Schema names cannot be used to retrieve metadata against an ANSI-compliant database. Set the driver compatibility parameter "Supports Schemas in Metadata Retrieval?" on page 80 to Boolean False. The database objects available for metadata retrieval are those visible to the database user who authenticated to the database. Schema qualifiers cannot be used to identify database objects. Therefore, to avoid naming collisions (such as owner1.table1, owner2.table1), give the database authentication user only SELECT privileges on objects being synchronized.
- When used against ANSI-compliant databases, usernames must be in uppercase. Set the driver compatibility parameter "Force Username Case" on page 78 to `upper`.

Microsoft JDBC Driver for SQL Server

- "Driver Information" on page 188
- "Compatibility" on page 189
- "Security" on page 189
- "URL Properties" on page 189

Driver Information

Table 14-10 *Microsoft JDBC Driver for SQL Server Settings*

Supported Database Versions:	Microsoft SQL Server 2022, Microsoft SQL Server 2019, Microsoft SQL Server 2016 Service Pack 1
Class Name	com.microsoft.sqlserver.jdbc.SQLServerDriver
Type	4
URL Syntax	jdbc:sqlserver://<host>[:<port1433>];databaseName=<database>
Download Instructions	Microsoft JDBC Driver for SQL Server
Filenames	mssql-jdbc-9.4.0.jre8.jar for 4.8.6, mssql-jdbc-7.2.2.jre11.jar for 4.8.7

Compatibility

The Microsoft JDBC driver works with all supported versions of Microsoft SQL Server.

Security

The Microsoft JDBC driver supports encrypted transport.

NOTE: For more information on SSL encryption, see Connection String Options page.

URL Properties

Delimit URL properties by using a semicolon (;).

The following table lists values for the integratedSecurity URL property for JDBC driver for SQL Server.

Table 14-11 *Values for the integratedSecurity URL Property*

Legal Value	Description
`false`	The default value. JDBC authentication is used.
`true`	Windows process-level authentication is used.

jTDS JDBC Driver

- "Driver Information" on page 190
- "Limitations" on page 190
- "Compatibility" on page 190
- "Security" on page 190
- "URL Properties" on page 190

Driver Information

Table 14-12 *jTDS Driver Settings*

Supported Database Versions:	Microsoft SQL Server 2005 Service Pack 1 or later, 2008 and 2008 R2, Sybase Adaptive Server Enterprise (ASE) 15 or later
Class Name	net.sourceforge.jtds.jdbc.Driver
Type	4 (2 if NTLM or SSO authentication is enabled)
URL Syntax	*jdbc:jtds:sqlserver://ip-address/*database-name
	*jdbc:jtds:sybase://ip-address/*database-name
Download Instructions	The jTDS Project (http://jtds.sourceforge.net/)
Filenames	`jtds-<version>.jar`

Limitations

The jTDS JDBC driver does not support views or stored procedures. We recommend you to use the Microsoft 2000 JDBC driver when Subscribing to views.

Compatibility

The jTDS driver works with all versions of Microsoft SQL Server. It also supports all versions of Sybase ASE, but it has not been tested by OpenText against that database server yet. Driver updates are infrequent.

Security

The jTDS driver supports encrypted transport.

URL Properties

Delimit URL properties by using a semicolon (;).

The following table lists values for the domain URL property for the jTDS driver.

Table 14-13 *Values for the Domain URL Property*

Legal Value	Description
<any-domain-name>	When a domain name is specified, either NTLM or SSO authentication can be used. NTLM authentication is selected when a username and password are supplied. SSO authentication is selected when a username and password are not supplied.
<no-value>	JDBC authentication is used.

The following table lists values for the SSL URL property for the jTDS driver.

Table 14-14 Values for the SSL URL Property

Legal Value	Description
`off`	SSL is not used. This is the default.
`request`	SSL is used if the server supports it.
`require`	SSL is required. An exception is thrown if the server doesn't support it.
`authenticate`	SSL is required. An exception is thrown if the server doesn't support it. In addition, the server's certificate must be signed by a trusted certificate authority.

MySQL Connector/J JDBC Driver

- "Driver Information" on page 191
- "Compatibility" on page 191
- "Security" on page 192
- "Required Parameter Settings for MyISAM Tables" on page 192

Driver Information

Table 14-15 MySQL Connector/J JDBC Driver Settings

Supported Database Versions	8 or later
Class Name	com.mysql.cj.jdbc.Driver
Type	4
URL Syntax	jdbc:mysql://*ip-address*:3306/*database-name*
Download Instructions	Download and extract. The `jar` file is located in the *extract-dir*/mysql-connector-java-*version* directory. MySQL Connector/J (http://www.mysql.com/products/connector/j/)
Filename	`mysql-connector-java-`*`version`*`-bin.jar`, `mysql-connector-j-8.0.33.jar`
Documentation URLs	For more information on MySQL Connector/J Documentation, see MySQL Refereal manual (http://dev.mysql.com/doc/refman/5.0/en/) For more information on Connecting Over SSL, see MySQL Refereal manual (http://dev.mysql.com/doc/refman/5.0/en/)

Also see "Generation/Retrieval Method (Table-Global)" on page 85.

Compatibility

The Connector/J driver is backward compatible. Database server updates are frequent. Driver updates are infrequent.

Security

The Connector/J driver supports JSSE (Java Secure Sockets Extension) SSL-encrypted transport.

Required Parameter Settings for MyISAM Tables

The following table lists driver parameters that you must set so that the JDBC driver can interoperate with the Connector/J driver against MyISAM tables.

Table 14-16 *Settings for MyISAM Tables*

Display Name	Tag Name	Value
Use manual transactions?	use-manual-transactions	false

Oracle Thin Client JDBC Driver

- "Driver Information" on page 192
- "Compatibility" on page 193
- "Security" on page 193
- "Dynamic Parameter Defaults" on page 193
- "Connection Properties" on page 193
- "Known Issues" on page 193

Driver Information

Table 14-17 *Oracle Thin Client Settings*

Supported Database Versions	19c and 21c
Class Name	oracle.jdbc.driver.OracleDriver
Type	4
URL Syntax	sid - jdbc:oracle:thin:@*ip-address*:1521:*<sid>*
	cdb - jdbc:oracle:thin:@*ip-address*:1521/<service>
Filenames	For 18c and 19c: `ojdbc8.jar` available in Oracle Download site.
Documentation URLs	For more information on Oracle Advanced Security, see Stanford University (http://www.stanford.edu/) page
	For more information on JDBC FAQ, see Oracle Technology (http://www.oracle.com/technology/) page.

Compatibility

The Thin Client driver is backward compatible. Database server updates and driver updates are infrequent.

Oracle releases thin client drivers for various JVMs. Even though all of them work with this product, we recommend that you use the 1.6 version.

Security

The Thin Client driver supports Oracle Advanced Security encrypted transport.

Dynamic Parameter Defaults

The following table lists driver compatibility parameters that the JDBC driver implicitly sets at runtime. Do not explicitly override these settings.

Table 14-18 *Oracle Thin Client Settings Not to Override*

Display Name	Tag Name	Value
Number of returned result sets:	handle-stmt-results	single

Connection Properties

The following table lists important connection properties for this driver.

Table 14-19 *Oracle Thin Client: Connection Properties*

Property	Significance
includeSynonyms	If the value of this property is `true`, synonym column metadata is available.
ORACLE.NET.ENCRYPTION_CLIENT	Defines the level of security that the client wants to negotiate with the server.
ORACLE.NET.ENCRYPTION_TYPES_CLIENT	Defines the encryption algorithm to be used.
ORACLE.NET.CRYPTO_CHECKSUM_CLIENT	Defines the level of security that it wants to negotiate with the server for data integrity.
ORACLE.NET.CRYPTO_CHEKSUM_TYPES_CLIENT	Defines the data integrity algorithm to be used.

Known Issues

- High CPU utilization triggered by execution of embedded SQL statements:

The most common problem experienced with this driver is high CPU utilitization. As a result, this driver always indicates that more results are available from calls to method `java.sql.Statement.execute(String stmt)`, which can lead to an infinite loop condition. This condition occurs only if all the following happen:

- A value other than `single`, no, or one in the driver compatibility parameter "Number of Returned Result Sets" on page 75 is being executed.
- An embedded SQL statement is being executed.
- The type of statement is not explicitly specified.

To avoid the conditions that produce high CPU utilization:

- Do not explicitly set this parameter. Defer to the dynamic default value.
- Always place a `jdbc:type` attribute on embedded `<jdbc:statement>` elements.

  NOTE: The jdbc namespace prefix must map to `urn:dirxml:jdbc`.

- Can't retrieve synonym column metadata.

  The connection property includeSynonyms must be set to `true`.

- Can't see synonym table primary key constraint.

  The only known solution to this problem is to use a view.

Oracle OCI JDBC Driver

Table 14-20 *Oracle OCI JDBC Driver Settings*

Supported Database Versions	12c, 18c, and 19c
Class Name	oracle.jdbc.driver.OracleDriver
Type	2
URL Syntax	jdbc:oracle:oci8:*@tns-name*
Filenames	For 12c: `ojdbc7.jar` available in Oracle Download site. For 18c and 19c: `ojdbc8.jar` available in Oracle Download site.
Documentation URLs	Oracle Call Interface (http://www.oracle.com/technology/) OCI FAQ (http://www.oracle.com/technology/) Oracle Advanced Security (http://www.stanford.edu/) Instant Client (http://www.oracle.com/technology/tech/oci/instantclient/index.html) Instant Client (https://docs.oracle.com/cd/B19306_01/server.102/b14357/ape.htm)

You can install SQLNet by doing either of the following:

- Use the Instant Client, which bypasses unneeded components of the full version.
- Download the full package from Oracle.

If the database is running on the same server as Identity Manager, you don't need to install SQLNet because SQLNet comes as standard on the database server.

The Oracle OCI driver is essentially the same as the Thin Client driver. See "Oracle Thin Client JDBC Driver" on page 192. The OCI client differs in the following ways:

- The OCI Client supports clustering, failover, and high availability.
- The OCI Client has additional security options.

For information on setting up the Oracle OCI Client, see Appendix M, "Setting Up an OCI Client on Linux," on page 245.

PostgreSQL JDBC Driver

- "Driver Information" on page 195
- "Compatibility" on page 195
- "Security" on page 196

Driver Information

Table 14-21 *PostgreSQL JDBC Driver Settings*

Supported Database Versions	12, 13, 14 15 and 16
Class Name	org.postgresql.Driver
Type	4
URL Syntax	jdbc:postgresql://*ip-address*:5432/*database-name*
Download Instructions	JDBC Driver Download (http://jdbc.postgresql.org/download.html)
Filename	postgresql-42.6.1.jar
Documentation URLs	JDBC Driver Documentation (http://jdbc.postgresql.org/)
	Using SSL (http://jdbc.postgresql.org/documentation/80/ssl.html)

The filename of the PostgreSQL varies by database version.

Compatibility

The latest builds of the PostgreSQL driver are backward compatible through server version 8.4.3. Database server updates and driver updates are frequent.

Security

The PostgreSQL driver supports SSL-encrypted transport for JDBC 3 driver versions.

Sybase Adaptive Server Enterprise JConnect JDBC Driver

- "Driver Information" on page 196
- "Compatibility" on page 196
- "Security" on page 196
- "Connection Properties" on page 196

Driver Information

Table 14-22 *Sybase Adaptive Server Enterprise Driver Settings*

Supported Database Versions	Adaptive Server Enterprise 15.0 or later
Class Name	com.sybase.jdbc3.jdbc.SybDriver (for jconn3.jar) com.sybase.jdbc4.jdbc.SybDriver (for jconn4.jar)
Type	4
URL Syntax	jdbc:sybase:Tds:*ip-address*:2048/*database-name*
Download Instructions	Sybase Downloads (http://www.sybase.com/downloads)
Filenames	`jconn3.jar` or `jconn4.jar`
Documentation URLs	For more information on jConnect Documentation, see Sybase inforcenter page.

For JDBC 3 (Java 1.4) versions and later.

Compatibility

The Adaptive Server driver is backward compatible. Database server updates and driver updates are infrequent.

Security

The Adaptive Server driver supports SSL-encrypted transport. To enable SSL encryption, you must specify a custom socket implementation via the SYBSOCKET_FACTORY connection property. For additional information on how to set connection properties, see "Connection Properties" on page 70.

Connection Properties

The SYBSOCKET_FACTORY property can be used to specify the class name of a custom socket implementation that supports encrypted transport.

MariaDB Connector/J JDBC Driver

- "Driver Information" on page 197
- "Security" on page 197
- "Required Parameter Settings for MyISAM Tables" on page 197

Driver Information

Table 14-23 *MariaDB Connector/J JDBC Driver Settings*

Supported Database Versions	10.2.13
Class Name	org.mariadb.jdbc.Driver
Type	4
URL Syntax	jdbc:mysql://*ip-address*:3306/*database-name*
Download Instructions	Download and extract. The `jar` file is located in the *extract-dir*/mysql-connector-java-*version* directory. MariaDB Connector (https://downloads.mariadb.org/connector-java/2.2.2/)
Filename	`mariadb-java-client-`*`version`*`.jar`
Documentation URLs	For more information on MariaDB Connector/J Documentation, see MariaDB Connector/J Referral manual (https://mariadb.com/kb/en/library/about-mariadb-connector-j/)

Also see "Generation/Retrieval Method (Table-Global)" on page 85.

Security

The Connector/J driver supports JSSE (Java Secure Sockets Extension) SSL-encrypted transport.

Required Parameter Settings for MyISAM Tables

The following table lists driver parameters that you must set so that the JDBC driver can interoperate with the Connector/J driver against MyISAM tables.

Table 14-24 *Settings for MyISAM Tables*

Display Name	Tag Name	Value
Use manual transactions?	use-manual-transactions	false

Supported Third-Party JDBC Drivers (Not Recommended)

This section identifies third-party JDBC drivers that are supported but whose use with the Identity Manager JDBC driver is not recommended:

- "Third-Party JDBC Driver Features" on page 198
- "JDBC URL Syntaxes" on page 198
- "JDBC Driver Class Names" on page 199
- "IBM DB2 Universal Database JDBC Driver" on page 199
- "Microsoft SQL Server 2008 JDBC Driver" on page 200
- "Microsoft SQL Server 2008 R2 JDBC Driver" on page 201

For information about supported third-party drivers that are recommended, see "Supported Third-Party JDBC Drivers (Not Recommended)" on page 198.

Third-Party JDBC Driver Features

The following table summarizes third-party JDBC driver features:

Table 14-25 *Third-Party JDBC Driver Features*

Driver	Supports Encrypted Transport?	Supports Retrieval of Auto-Generated Keys?
IBM DB2 UDB Type 3	No	No
Microsoft 2008	Yes	Yes
Microsoft 2008 R2	Yes	Yes

JDBC URL Syntaxes

The following table lists URL syntaxes for supported third-party JDBC drivers:

Table 14-26 *URL Syntaxes*

Third-Party JDBC Driver	JDBC URL Syntax
IBM DB2 UDB Type 3	jdbc:db2:*//ip-address*:6789/*database-name*
Microsoft SQL Server 2008	jdbc:sqlserver:*//ip-address-or-dns-name*:1433;databaseName=*database-name*
Microsoft SQL Server 2008 R2	jdbc:sqlserver:*//ip-address-or-dns-name*:1433;databaseName=*database-name*

This information is used in conjunction with the Authentication Context parameter. For information on this parameter, see "Authentication Context" on page 57.

JDBC Driver Class Names

The following table lists the fully-qualified Java class names of supported third-party JDBC drivers:

Table 14-27 *Class Names of Third-Party JDBC Drivers*

Third-Party JDBC Driver	Class Name
IBM DB2 UDB Type 3	COM.ibm.db2.jdbc.net.DB2Driver
Microsoft 2008	com.microsoft.sqlserver.jdbc.SQLServerDriver
Microsoft 2008 R2	com.microsoft.sqlserver.jdbc.SQLServerDriver

This information is used in conjunction with the JDBC Driver Class Name parameter. For information on this parameter, see "Third-Party JDBC Driver Class Name" on page 60.

IBM DB2 Universal Database JDBC Driver

- "Driver Information" on page 199
- "Compatibility" on page 200
- "Security" on page 200
- "Known Issues" on page 200

Driver Information

Table 14-28 *IBM DB2 Driver Settings*

	Type 4 Driver
Supported Database Versions	9.x
Class Name	com.ibm.db2.jcc.DB2Driver
URL Syntax	`jdbc:db2://ip-address:50000/`*database-name*
Download Instructions	Copy the file from the database server. `file:///database-installation-directory/java`
File Name	`db2jcc.jar`
Documentation URLs	DB2 Information Center (http://publib.boulder.ibm.com/infocenter/db2v7luw) JDBC Programming (http://publib.boulder.ibm.com/infocenter/db2v7luw/index.jsp?topic=/com.ibm.db2v7.doc/db2a0/db2a0159.htm)

Compatibility

The IBM DB2 driver can best be characterized as version-hypersensitive. It is not compatible across major or minor versions of DB2, including FixPacks. For this reason, we recommend that you use the file installed on the database server.

The IBM DB2 driver must be updated on the Identity Manager or Remote Loader server every time the target database is updated, even if only at the FixPack level.

Security

The IBM DB2 driver does not support encrypted transport.

Known Issues

- A version mismatch usually results in connectivity-related failures.

  The most common problem experienced with the IBM DB2 driver is because of a driver/database version mismatch. The symptom of a version mismatch is connectivity-related failures such as `CLI0601E Invalid statement handle or statement is closed`. To remedy the problem, overwrite the `db2java.zip` file on the Identity Manager or Remote Loader server with the version installed on the database server.

- It's very difficult to diagnose and remedy Java-related errors on the database server.

  Numerous error conditions and error-codes can arise when you attempt to install and execute user-defined stored procedures and functions written in Java. Diagnosing them can be time consuming and frustrating. A log file (`db2diag.log` on the database server) can often provide additional debugging information. In addition, all error codes are documented and available online.

Microsoft SQL Server 2008 JDBC Driver

- "Driver Information" on page 201
- "Compatibility" on page 201
- "Security" on page 201
- "URL Properties" on page 201

Driver Information

Table 14-29 *Microsoft SQL Server 2008 Driver Settings*

Supported Database Versions	2008
Class Name	com.microsoft.sqlserver.jdbc.SQLServerDriver
Type	4 (2 if integrated security is enabled)
URL Syntax	jdbc:sqlserver://*ip-address-or-dns-name*:1433;databaseName=*database-name*
Download Instructions	Microsoft SQL Server 2008 JDBC Driver (http://msdn2.microsoft.com/en-us/data/aa937724.aspx)
Filenames	`sqljdbc.jar`

The filename, URL syntax, and classname differ (often subtly) from those of the 2000 driver.

Compatibility

The SQL Server 2008 driver works only with SQL Server 2005. Database server and driver updates are infrequent.

Security

The SQL Server 2008 driver supports encrypted transport.

URL Properties

Delimit URL properties by using a semicolon (;).

The following table lists values for the integratedSecurity URL property for the SQL Server 2005 driver.

Table 14-30 *Values for the integratedSecurity URL Property*

Legal Value	Description
`false`	The default value. JDBC authentication is used.
`true`	Windows process-level authentication is used.

Microsoft SQL Server 2008 R2 JDBC Driver

- "Driver Information" on page 202
- "Compatibility" on page 202
- "Security" on page 202
- "URL Properties" on page 202

Driver Information

Table 14-31 Microsoft SQL Server 2008 Driver Settings

Supported Database Versions:	2008 R2
Class Name	com.microsoft.sqlserver.jdbc.SQLServerDriver
Type	4 (2 if integrated security is enabled)
URL Syntax	jdbc:sqlserver://*ip-address-or-dns-name*:1433;databaseName=*database-name*
Download Instructions	Microsoft SQL Server 2008 JDBC Driver (http://msdn2.microsoft.com/en-us/data/aa937724.aspx)
Filenames	`sqljdbc.jar`

The filename, URL syntax, and classname differ (often subtly) from those of the 2000 driver.

Compatibility

The SQL Server 2008 R2 driver works only with SQL Server 2008. Database server and driver updates are infrequent.

Security

The SQL Server 2008 driver supports encrypted transport.

URL Properties

Delimit URL properties by using a semicolon (;).

The following table lists values for the integratedSecurity URL property for the SQL Server 2008 driver.

Table 14-32 Values for the integratedSecurity URL Property

Legal Value	Description
`false`	The default value. JDBC authentication is used.
`true`	Windows process-level authentication is used.

Deprecated Third-Party JDBC Drivers

The BEA WebLogic jDriver for Microsoft SQL Server is no longer supported.

Other Third-Party JDBC Drivers

This section lists an unsupported driver of interest (IBM Toolbox for Java/JTOpen) and discusses how to assess the feasibility of using unsupported third-party JDBC drivers with this product.

- "IBM Toolbox for Java/JTOpen" on page 203
- "Minimum Third-Party JDBC Driver Requirements" on page 203
- "Considerations When Using Other Third-Party JDBC Drivers" on page 204

IBM Toolbox for Java/JTOpen

Table 14-33 *Settings for IBM Toolbox for Java/JTOpen*

Database	IBM Toolbox for Java/JTOpen ◆ iSeries Toolbox for Java (alias) ◆ AS/400 Toolbox for Java (alias)
Class Name	com.ibm.as400.access.AS400JDBCDriver
Type	4
URL Syntax	jdbc:as400:`//ip-address/database-name`
Download Instructions	Download URLs for JTOpen ◆ JTOpen (http://jt400.sourceforge.net) ◆ Toolbox for Java/JTOpen (http://www-03.ibm.com/servers/eserver/iseries/toolbox/downloads.html)
Filenames	`jt400.jar`
Documentation URLs	Toolbox for Java/JTOpen (http://www-03.ibm.com/servers/eserver/iseries/toolbox/)

If you use the IBM Toolbox for Java/JTOpen driver, you must manually enter values for the JDBC Driver Class Name and Authentication Context parameters. The settings are not automatically populated. See "Third-Party JDBC Driver Class Name" on page 60 and "Authentication Context" on page 57.

Minimum Third-Party JDBC Driver Requirements

The JDBC driver might not interoperate with all third-party JDBC drivers. If you use an unsupported third-party JDBC driver, it must meet the following requirements:

- Support required metadata methods

  For a current list of the required and optional java.sql.DatabaseMetaData method calls that the JDBC driver makes, see Appendix F, "java.sql.DatabaseMetaData Methods," on page 227.

- Support other required JDBC methods

  For a list of required JDBC methods that the JDBC driver uses, refer to Appendix G, "JDBC Interface Methods," on page 229. You can use this list in collaboration with third-party driver documentation to identify potential incompatibilities.

Considerations When Using Other Third-Party JDBC Drivers

- Because the JDBC driver is directly dependent upon third-party JDBC driver implementations, bugs in those implementations might cause this product to malfunction.

  To assist you in debugging third-party JDBC drivers, the JDBC driver supports the following:

  - Tracing at the JDBC API level (level 6)
  - Third-party JDBC driver (level 7) tracing
- Stored procedure or function support is a likely point of failure.
- You probably need to write a custom driver descriptor file.

  Specifically, you need to categorize error codes and SQL states for the third-party driver that you are using.

Security Issues

To ensure that a secure connection exists between the Identity Manager JDBC driver and a third-party driver, we recommend the following:

- Run the JDBC driver remotely on the database server.
- Use SSL to encrypt communications between the Identity Manager server and the database server.

If you cannot run the JDBC driver remotely, you might want to use a type 2 or type 3 JDBC driver. These driver types often facilitate a greater degree of security through middleware servers or client APIs unavailable to other JDBC driver types. Some type 4 drivers support encrypted transport, but encryption is the exception rather than the rule.

15 Troubleshooting the JDBC Driver

- "DirXML-Accounts Attribute Shows Incorrect Value When a User is Enabled or Disabled in the OpenText Identity Vault on DB2 and Oracle Database Drivers" on page 205
- "Password Changes for Users Are Not Synchronized from the OpenText Identity Vault for the Oracle Database Driver" on page 205
- "Modifying the Default Configuration for the Sybase Driver in Direct Mode" on page 206
- "Driver Shim Is Irresponsive When a Connected Database Server Does Not Respond" on page 206
- "Recognizing Publication Events" on page 206
- "Executing Test Scripts" on page 207
- "Applying the Latest Driver Package Does Not Change the Default Setting of Enable Service Channel ECV" on page 207
- "Troubleshooting Driver Processes" on page 207
- "Null Pointer Exception Appears in the Trace for JDBC Driver for Connecting to Oracle Database While Using ojdbc10.jar" on page 208

DirXML-Accounts Attribute Shows Incorrect Value When a User is Enabled or Disabled in the OpenText Identity Vault on DB2 and Oracle Database Drivers

This issue is observed with certain settings on the driver:

DB2 driver: When the **Allow Loopback** option is set to **Yes** on the Publisher channel in the **Indirect Triggered** mode.

Workaround: Disable the **Allow Loopback** option.

Oracle Driver: In the **Direct Triggerless** mode.

There is no workaround.

Password Changes for Users Are Not Synchronized from the OpenText Identity Vault for the Oracle Database Driver

The Oracle Database driver created with an Indirect/Direct Synchronization sample package used an incorrect query to create or change the name and password of the users. The driver patch 2 fixed this issue, but after upgrading to patch 2, if the password is reset for an existing user, it fails with an error and displays the following message:

```
User does not exist.
```

To workaround this issue, after upgrading the driver, delete the existing user from the Oracle database and synchronize it from the Identity Vault.

Modifying the Default Configuration for the Sybase Driver in Direct Mode

When you add users using the default configuration on the subscriber channel for the Sybase driver in direct mode, an error is displayed. Change the driver settings as follows:

- Set the **Generation/retrieval method (table-global)** to **Subscriber-generated**.
- Set the **Retrieval timing (table-global)** to **after row insertion**.
- Leave the **Method and timing (table-local)** as **blank**

When you change the value for the **Method and timing** option, you need to edit the sample procedures appropriately. For example, if you set it to **view_usr("indirect.proc_idu(pk_idu)"); view_grp("indirect.proc_idg(pk_idg)")**, you must edit the `indirect.proc_idu` and `indirect.proc_idg` procedures so that unique values are returned for the **idg** and **idu** columns respectively.

Driver Shim Is Irresponsive When a Connected Database Server Does Not Respond

This issue is observed in the following cases:

- **Case 1:** The driver tries to update a record that is currently being modified through another database connection.

  If you instruct the driver to update this record, the driver will wait to operate on it until the updated record is committed to the database.

  This is an expected behavior. The driver is designed to wait to eliminate the chances of inconsistent results in such situations.
- **Case 2:** When the transaction log in a database is filled. For example, Sybase database.

  The database does not allow any operations until the transaction log is cleared.

  It is the responsibility of the database administrator to clear the transactions log. When the log is cleared, the database is unlocked for operations and the driver resumes processing without issues.

Recognizing Publication Events

Publication events might not be recognized by the Publisher channel unless you explicitly commit changes. For the commit keywords of supported databases, see "Commit Keywords" on page 172.

Executing Test Scripts

The test scripts should be executed by a user other than the driver's `idm` database user account. If you execute them as the `idm` user, events are ignored by the driver's Publisher channel, unless publication loopback is allowed. For additional information on allowing or disallowing publication loopback, refer to "Allow Loopback?" on page 95.

Applying the Latest Driver Package Does Not Change the Default Setting of Enable Service Channel ECV

Issue: If you upgraded to Identity Manager 4.7 and updated the base packages for your driver, the package update process does not overwrite the default setting (`False`) of **Enable Service Channel** ECV.

This issue does not occur when you create a new driver.

Workaround: Manually change the ECV for the driver.

To change the ECV in Designer:

1. In Modeler, right-click the driver line.
2. Select **Properties > Engine Control Values**.
3. Click the tooltip icon to the right of **Engine Controls for Server**.

   If a server is associated with the Identity Vault, and if you are authenticated, the engine control values display in the large pane.
4. Change the value for **Enable Subscriber Service Channel**.
5. Click **OK**.
6. For the change to take effect, deploy the driver to the live Identity Vault.

To change the ECV in Identity Console:

1. Click the **IDM Administration** tile.
2. On the Driver Dashboard, locate the driver icon and click the icon to display the driver's properties page.
3. Click the **Configuration** tab.
4. Expand the **Engine Control Values** section.
5. Change the value for **Enable Subscriber Service Channel**.
6. Save changes.
7. Restart the driver for the changes to take effect.

Troubleshooting Driver Processes

Viewing driver processes is necessary to analyze unexpected behavior. To view the driver processing events, use DSTrace. You should only use it during testing and troubleshooting the driver. Running DSTrace while the drivers are in production increases the utilization on the Identity Manager server and can cause events to process very slowly. For more information, see "Viewing Identity Manager

Processes (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/b1rc1vm.html)" in the *OpenText™ Identity Manager CE 24.4 (v4.10) Driver Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/bktitle.html)*.

Null Pointer Exception Appears in the Trace for JDBC Driver for Connecting to Oracle Database While Using ojdbc10.jar

Issue: The `ojdbc10.jar` is not supported for the JDBC Driver.

Fix: For the driver to work you must use:

- `ojdbc8.jar` for Oracle 18c and 19c
- `ojdbc7.jar` for Oracle 12c

For more information about the `jar` file to be used refer to "Oracle Thin Client JDBC Driver" on page 192.

A Uninstalling the Driver

- "Deleting Identity Manager Driver Objects" on page 209
- "Running the Product Uninstaller" on page 209
- "Executing Database Uninstallation Scripts" on page 209

We recommend you to install and uninstall preconfigured drivers and database scripts as a unit. To prevent unintentional mismatching, database scripts and preconfigured drivers contain headers with a version number, the target database name, and the database version.

Deleting Identity Manager Driver Objects

When you are deleting NetIQ Identity Vault objects, you must delete all child objects before you can delete a parent object. For example, you must delete all rules and style sheets on the Publisher channel before you can delete the Publisher object. Similarly, you must delete both the Publisher and Subscriber objects before you can delete the Driver object.

To remove a driver object from an Identity Vault:

1. In Identity Console, click the **IDM Administration** tile.
2. On the Driver Dashboard, search for the driver you want to delete.
3. Click the upper right corner of the driver, click **Delete Driver**.

Running the Product Uninstaller

Uninstallation procedures vary by platform.

To uninstall the Identity Manager JDBC driver on Windows, use **Add or Remove Programs** in the **Control Panel**.

Executing Database Uninstallation Scripts

This section provides helps you execute database uninstallation SQL scripts.

- "IBM DB2 Universal Database (UDB) Uninstallation" on page 210
- "Informix Dynamic Server (IDS) Uninstallation" on page 210
- "Microsoft SQL Server Uninstallation" on page 210
- "MySQL Uninstallation" on page 211
- "Oracle Uninstallation" on page 211
- "PostgreSQL Uninstallation" on page 211
- "Sybase Adaptive Server Enterprise (ASE) Uninstallation" on page 212

IBM DB2 Universal Database (UDB) Uninstallation

The directory context for DB2 is *install-dir*\DirXMLUtilities\jdbc\sql\db2_udbl\install.

1. Drop the idm, indirect, and direct operating system user accounts.
2. If you haven't already done so, change the name of the administrator account name and password in the installation scripts.
3. Using the Command Line Processor (CLP), execute the uninstall.sql script.

   For example: db2 -f uninstall.sql

   This script won't execute in the Command Center interface beyond version 7 because the script uses the \ line continuation character. Later versions of the Command Center don't recognize this character.
4. Delete the idm_db2.jar file.

Informix Dynamic Server (IDS) Uninstallation

The directory context for Informix SQL scripts is *install-dir*\DirXMLUtilities\jdbc\sql\informix_ids\install.

1. Drop the idm operating system user account.
2. Start a client such as SQL Editor.
3. Log on to your server as user informix or another user with DBA (database administrator) privileges.

   By default, the password for informix is informix.

   If you execute scripts as a user other than informix, change all references to informix in the install scripts prior to execution.
4. If you aren't using the informix account with the default password, change the name of the DBA account name and password in the installation scripts.
5. Open and execute uninstall.sql from the ansi (transactional, ANSI-compliant), log (transactional, non-ANSI-compliant), or no_log (non-transactional, non-ANSI-compliant) subdirectory, depending upon which type of database you installed.

Microsoft SQL Server Uninstallation

The directory context for Microsoft SQL Server scripts is *install-dir*\DirXMLUtilities\jdbc\sql\mssql\install.

1. Start a client such as Query Analyzer.
2. Log on to your database server as user sa.

   By default, the sa user has no password.
3. Open and execute the first installation script, uninstall.sql.

   The execute hotkey in Query Analyzer is F5.

MySQL Uninstallation

The directory context for MySQL SQL scripts is *install-dir*\DirXMLUtilities\jdbc\sql\mysql\install.

1. From a MySQL client, such as mysql, log on as user root or another user with administrative privileges.

   For example, from the command line execute mysql -u root -p

   By default, the root user has no password.

2. Execute the uninstall.sql uninstallation script.

   For example: mysql> \. c:\uninstall.sql

   Don't use a semicolon to terminate this statement.

Oracle Uninstallation

The directory context for Oracle SQL scripts is *install-dir*\DirXMLUtilities\jdbc\sql\oracle\install.

1. From an Oracle client, such as SQL Plus, log on as user SYSTEM.

   By default, the password for SYSTEM is MANAGER.

   If you execute scripts as a user other than SYSTEM with password MANAGER, change all references to SYSTEM in the scripts prior to execution.

2. Execute the uninstallation script uninstall.sql.

   For example: SQL> @c:\uninstall.sql

PostgreSQL Uninstallation

The directory context for PostgreSQL scripts is *install-dir*\DirXMLUtilities\jdbc\sql\postgres\install. The directory context for executing Postgres commands is *postgres-install-dir*/pgsql/bin.

1. From a Postgres client such as psql, log on as user postgres to the idm database.

   For example, from the UNIXC command line, execute ./psql -d idm postgres

   By default, the Postgres user has no password.

2. From inside psql, execute the script uninstall.sql.

   For example: idm=# \i uninstall.sql

3. Drop the database idm.

   For example, from the UNIX command line, execute ./dropdb idm

4. Remove or comment out entries for the idm user in the pg_hba.conf file.

   For example:

   ```
   #host     idm          idm     255.255.255.255   255.255.255.0
   ```

5. Restart the Postgres server to effect changes made to the pg_hba.conf file.

Sybase Adaptive Server Enterprise (ASE) Uninstallation

The directory context for Sybase SQL scripts is *install-dir*\DirXMLUtilities\jdbc\sql\sybase_ase\install.

1. From a Sybase client, such as isql, log on as user sa.
2. Execute the installation script uninstall.sql.

   For example, from the command line, execute isql -U sa -P -i uninstall.sql

   By default, the sa account has no password.

B Known Issues and Limitations

OpenText strives to ensure that our products provide quality solutions for your enterprise software needs. The following issues are currently being researched. If you need further assistance with any issue, please contact Technical Support.

For the list of the known issues in Identity Manager 4.5, Identity Manager Standard Edition 4.5, and Identity Manager 4.5.1, see the Release Notes for each version on the Identity Manager Documentation page.

- "Known Issues" on page 213
- "Limitations" on page 214

Known Issues

JDBC Driver

- OpenText Identity Vault Time and Timestamp syntaxes are inadequate for expressing the range and granularity of their database counterparts. This is a publication problem because database time-related types typically have a wider range and greater degree of granularity (typically nanoseconds). The converse is not true. For more information, see "Time Syntax" on page 60.
- The JDBC driver is unable to parse proprietary database time stamp formats. Some databases, such as Sybase and DB2, have proprietary time stamp formats that the java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html) class can't parse. When synchronizing time stamp columns from these databases, the JDBC driver, by default, assumes that time stamp values placed in the event log table are in ODBC canonical format (that is, `yyyy-mm-dd hh:mm:ss.fffffffff`). The recommended method for enabling the JDBC driver to handle proprietary database time stamp formats is to implement a custom `DBTimestampTranslator` class. This interface is documented in the JavaDoc Tool that ships with the JDBC driver. Using this approach avoids the problem of reformatting time stamps in the database before they are inserted into the event log table or reformatted in style sheets. The JDBC driver ships with default implementations for the native DB2 time stamp format and the Sybase style 109 time stamp format.
- Statements executed against the database server might block indefinitely.

  Typically, blocking is caused by a database resource being exclusively locked. Because the locking mechanisms and locking SQL vary by database, the general solution to this problem is to implement a custom `DBLockStatementGenerator` class. For additional information, see "Lock Statement Generator Class" on page 76. The JDBC driver ships with a default implementation for Oracle.

  Many factors can cause blocking. To mitigate the likelihood of blocking, we recommend that you do not set the Transaction Isolation Level parameter to a level greater than `read committed`.

The JDBC interface defines a method java.sql.Statement.setQueryTimeout(int):void (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) that allows a statement to time out after a specified number of seconds. Unfortunately, implementations of this method between third-party JDBC drivers range from not being implemented to having bugs. For this reason, this method was deemed unsuitable as a general-purpose solution.

- The JDBC displays an exception error “User is unassociated" when performing a Modify and/or Delete event with dest-dn. This is because the JDBC driver does not support CRUD operation.
- In MSSQL database you can only add new values to the view only if the view is created from a single base table. If multiple base tables are used to create the view, you cannot update the view.

JDBC Fan-Out Driver

- If the Cached Priority Synchronisation event is a modify of the Referential Attribute, the driver sends a Success status. However, the modified change is not updated in the target database.

Limitations

- The JDBC driver does not support the use of delimited (quoted) database identifiers (for example, “names with spaces”).
- JDBC 2 data types are not supported, with the exception of Large Object data types (LOBs) such as `CLOB` and `BLOB`.
- JDBC 3 data types are not supported.
- PostgreSQL does not support `<check-object-password>` events. Authentication is controlled by manually inserting entries into the `pg_hba.conf` file.

C Best Practices

The following section lists important best practices for using the JDBC driver. You can find additional information in Chapter 6, “Configuring the JDBC Driver,” on page 53.

Tips for Synchronizing Millions of User Records on the Publisher Channel

For successfully synchronizing millions of user records, change the following settings using Designer:

1. Click the driver set that contains this driver and change the Java Maximum heap size to 256 MB. This change will be applicable to all the drivers under this driver set.
2. Under the **Driver Settings** tab, change the **Show the compatibility parameters?** option to **show** to display the **Show backward compatibility parameters?** option. Change it to **show,** then change the **Enable the Table Referential attribute support?** to **No**.
3. Under the **Publisher Settings** tab, change the **Show polling-related parameters?** option to **show** to display the **Batch Size**, then change it to 128.

Schema Name Use Cases

In the Schema Name under schema-aware mode the driver qualifies the tables according to the following rules:

- Every table that contains a primary key constraint is considered as an object class. For example, the following `usr` table created by the SQL code is considered as an object class by the driver shim if the Schema Name is set to `indirect`.

  ```
  CREATE TABLE indirect.usr
  (
    idu         INTEGER  NOT NULL,
    fname       VARCHAR2(64),
    lname       CHAR(64),
    CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
  )
  ```

  The above table will contain two single-valued attributes; `fname` and `lname`. The driver will build its associations based on the value of the column referenced in the primary key constraint, which in this case is column `idu`.

- Every table that only has a `Foreign Key` constraint is considered as a multi-valued attribute of the class that holds the primary key pointed by the said foreign key constraint. In the following example, if the driver parameter, `Schema Name` is set to `indirect`, then the `usr_phone.phoneno` table is considered a multi-valued attribute belonging to the class `usr`.

  NOTE: The child table `usr_phone` only has a foreign key constraint to the parent table `usr`. The table `usr` is considered an object class since it has a primary key constraint.

```
CREATE TABLE indirect.usr
(
  idu         INTEGER  NOT NULL,
  fname       VARCHAR2(64),
  lname       CHAR(64),
  CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
)
CREATE TABLE indirect.usr_phone
(
  idu      INTEGER       NOT NULL,
  phoneno  VARCHAR2(64)  NOT NULL,
  CONSTRAINT fk_phone_idu FOREIGN KEY (idu)
    REFERENCES indirect.usr(idu) ON DELETE CASCADE
)
```

DN-type references between the two objects will have different requirements based on whether the reference points back to the same object class or to a different object class. DN-type references also change depending on whether the DN attribute is single valued or multi-valued.

- If the attribute is single-valued and points back to the same object class, then the table that contains that class will have a foreign key constraint to itself and uses a local column to store the value. An example is the `manager` attribute in eDirectory. In the example below, if the driver parameter Schema Name is set to `indirect`, then the attribute `manager` will be a DN-type attribute that points to another row inside the `usr` table.

```
CREATE TABLE indirect.usr
(
  idu         INTEGER  NOT NULL,
  fname       VARCHAR2(64),
  lname       CHAR(64),
  manager     INTEGER,
  CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
  CONSTRAINT fk_usr_manager FOREIGN KEY (manager)
    REFERENCES indirect.usr(idu) ON DELETE SET NULL
)
```

- If the attribute is multi-valued and points back to the same object class, then we need a child table with columns, each having its own foreign key constraint to the same parent table. An example is the `directReports` attribute in eDirectory. In the following example, if the driver parameter Schema Name is set to `indirect`, then the attribute `usr_directReports.repname` will be a DN-type attribute that points to one or more rows inside the `usr` table.

```
CREATE TABLE indirect.usr
(
  idu         INTEGER  NOT NULL,
  fname       VARCHAR2(64),
  lname       CHAR(64),
  CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
)
CREATE TABLE indirect.usr_directReports
(
  idu      INTEGER       NOT NULL,
  repname  VARCHAR2(64)  NOT NULL,
  CONSTRAINT fk_directReports_idu FOREIGN KEY (idu)
    REFERENCES indirect.usr(idu) ON DELETE CASCADE,
  CONSTRAINT fk_directReports_repname FOREIGN KEY (idu)
    REFERENCES indirect.usr(idu) ON DELETE CASCADE
)
```

- If the attribute is single-valued and points to a different object class, then the table that contains that class will have a foreign key constraint to the table for the other object class, and a local column to store that value. An example is the `Host Server` attribute in eDirectory. In the following example, if the driver parameter `Schema Name` is set to `indirect`, then the attribute `hostsrv` will be a DN-type attribute that points to a row inside the `server` table. In this example, both the `volume` table and the `server` table represent object classes.

```
CREATE TABLE indirect.volume
(
  idv         INTEGER  NOT NULL,
  vname       VARCHAR2(64),
  hostsrv     INTEGER,
  CONSTRAINT pk_volume_idv     PRIMARY KEY (idv),
  CONSTRAINT fk_volume_hostsrv FOREIGN KEY (ids)
    REFERENCES indirect.server(ids) ON DELETE SET NULL
)
CREATE TABLE indirect.server
(
  ids         INTEGER  NOT NULL,
  srvname     VARCHAR2(64),
  CONSTRAINT pk_server_ids PRIMARY KEY (ids)
)
```

- If the attribute is multi-valued and points to a different object class, then we need a child table with two columns, each having a foreign key constraint to a different parent table. An example is the `Group Membership` attribute in eDirectory. In the following example, if the driver

parameter Schema Name is set to `indirect`, then the attribute `usr_mbr_of.idu` will be a DN-type attribute that points to a row inside the `grp` table. In this example, both the `usr` table and the `grp` table represent object classes.

```
CREATE TABLE indirect.usr
(
  idu         INTEGER  NOT NULL,
  fname       VARCHAR2(64),
  lname       CHAR(64),
  CONSTRAINT pk_usr_idu     PRIMARY KEY (idu),
)
CREATE TABLE indirect.grp
(
  idg         INTEGER  NOT NULL,
  for_insert  INTEGER,
  CONSTRAINT pk_grp_idg PRIMARY KEY (idg)
)
CREATE TABLE indirect.usr_mbr_of
(
  idu  INTEGER  NOT NULL,
  idg  INTEGER  NOT NULL,
  CONSTRAINT fk_mbr_of_idu FOREIGN KEY (idu)
    REFERENCES indirect.usr(idu) ON DELETE CASCADE,
  CONSTRAINT fk_mbr_of_idg FOREIGN KEY (idg)
    REFERENCES indirect.grp(idg) ON DELETE CASCADE
)
```

Security/Performance:

- For performance and security reasons, run the driver remotely on the database server whenever possible. Be sure to enable SSL encryption between the Identity Vault and the Remote Loader service.
- You should enable SSL encryption for third-party drivers whenever the JDBC driver is not running remotely on the database server. For information on the security capabilities of supported third-party drivers, see Chapter 14, “Third-Party JDBC Drivers,” on page 181.
- In a production environment, turn off tracing.

Other:

- For direct synchronization, prefix one or more view column names with “pk_” (case-insensitive).
- For both direct and indirect synchronization, use different primary key column names between logical database classes.
- Delimit (double-quote) primary key values placed in the event log `table_key` field if they contain the following characters: , ; ' + = \ " < > This caution is usually an issue only if the primary key column is a binary type.
- When an Identity Vault is the authoritative source of primary key values, GUID rather than CN is recommended for use as a primary key. Unlike CN, GUID is single-valued and does not change.
- From publication triggers, omit foreign key columns that link child and parent tables.
- If primary key columns are static (they do not change), do not include them in publication triggers.

- Place the `jdbc:type="query"` attribute value on all embedded `SELECT` statements. Place the `jdbc:type="update"` attribute value on all embedded `INSERT`, `UPDATE` and `DELETE` statements.
- To avoid issues that arise when you run a sql query that has a reserved word as a column name, specify a fully qualified name for the column.

  For example, when you use **group** as a column name under **usr** table, **group** being is a sql keyword, your query might not be properly executed. To avoid this, specify a fully qualified name **such as usr.group** (`<Tablename>.<Columnname>`) for the column.
- By design, the driver doesn't allow primary keys on child tables in the Synch Schema mode. To comply with standard database best practices, if you add a primary key on all tables including the child tables containing multi-valued attributes, the driver doesn't work properly in this mode. You must operate the driver in the Synch Tables/Views mode to allow primary keys on child tables. The Synch Tables/Views mode prevents the driver from adding the child tables to the list of synchronized tables.

D FAQ

- "Can't See Tables or Views" on page 221
- "Synchronizing with Tables" on page 221
- "Processing Rows in the Event Log Table" on page 222
- "Managing Database User Accounts" on page 222
- "Synchronizing Large Data Types" on page 222
- "Slow Publication" on page 222
- "Synchronizing Multiple Classes" on page 223
- "Encrypted Transport" on page 223
- "Mapping Multivalue Attributes" on page 223
- "Synchronizing Garbage Strings" on page 223
- "Running Multiple JDBC Driver Instances" on page 224

Can't See Tables or Views

Question: Why can't the driver see my tables or views?

Answer: The driver is capable of synchronizing only tables that have explicit primary key constraints and views that contain one or more columns prefixed with "pk_" (case-insensitive). The driver uses these constraints to determine which fields to use when constructing associations. As such, the driver ignores any unconstrained tables. If you are trying to synchronize with tables or views that lack the necessary constraints, either add them or synchronize to intermediate tables with the required constraints.

Another possibility is that the driver lacks the necessary database privileges to see the tables. Usually, visibility is determined by the presence or absence of the SELECT privilege.

Synchronizing with Tables

Question: How do I synchronize with tables located in multiple schemas?

Answer: Do one of the following:

- Alias the tables into the synchronization schema.
- Synchronize to intermediate tables in the synchronization schema and move the data across schema boundaries.
- Use a view.
- Create a virtual schema by using the Table/View Names parameter.

  See "Table/View Names" on page 68.

Processing Rows in the Event Log Table

Question: Why isn't the driver processing rows in the Event Log Table?

Answer: Do the following:

1. Check the `perpetrator` field of the rows in question and make sure that the value is set to something other than the driver's database username.

   The Publisher channel checks the `perpetrator` field to detect loopback events if the Publisher channel Allow Loopback parameter is set to Boolean False (the default). See "Allow Loopback?" on page 95.

   When the Allow Loopback parameter is set to Boolean False, the Publisher channel ignores all records where the `perpetrator` field value is equal to the driver's database username. The driver's database username is specified by using the Authentication ID parameter. See "Authentication ID" on page 57.

2. Ensure that the record's `status` field is set to N (new).

   Records with `status` fields set to something other than N will not be processed.

3. Make sure to explicitly commit changes.

   Changes are often tentative until explicitly committed.

Managing Database User Accounts

Question: Can the driver manage database user accounts?

Answer: Yes. You can manage database accounts by using embedded SQL. For more information, see Chapter 12, "Embedded SQL Statements in XDS Events," on page 141.

Synchronizing Large Data Types

Question: Can the driver synchronize large binary and string data types?

Answer: Yes. Large binary and string data types can be subscribed and published. Publish large binary and string data types by using query-back event types. For additional information, see "Event Types" on page 132.

Slow Publication

Question: Why is publication slow?

Answer: If the event log table contains a large number of rows, index the table. Example indexes are provided in all database installation scripts. By using trace level 3, you can view the statements that the driver uses to maintain the event log.

You can further refine indexes in the installation scripts to enhance publication performance. Placing indexes in a different tablespace or physical disk than the event log table also enhances publication performance.

Furthermore, in a production environment, set the Delete Processed Rows parameter to Boolean False, unless processed rows are being periodically moved to another table. See "Delete Processed Rows?" on page 95.

Synchronizing Multiple Classes

Question: Can the driver synchronize multiple classes?

Answer: Yes. However, primary key column names must be unique between logical database classes. For example, if *class1* is mapped to *table1* with primary key column name *key1,* and *class2* is mapped to *table2* with primary key column name *key2*, then the name of *key1* cannot equal *key2*.

This requirement can always be satisfied, no matter which synchronization model is employed.

Encrypted Transport

Question: Does the driver support encrypted transport?

Answer: No. How the driver communicates with a given database depends upon the third-party driver being used. Some third-party drivers support encrypted transport, but others do not. Even if encrypted transport is supported, no standardized way exists to enable encryption between third-party JDBC drivers.

The general solution for this problem is to remotely run the JDBC driver and your third-party driver. This method allows both the JDBC driver and the third-party driver to run locally on the database server. Then all data traveling across the network between the Identity Manager engine and the JDBC driver are SSL encrypted.

Another possibility is to use a type 3 or type 2 third-party JDBC driver. Database middleware and client APIs usually provide encrypted transport mechanisms.

Mapping Multivalue Attributes

Question: How do I map multivalue attributes to single-value database fields?

Answer: See "Mapping Multivalue Attributes to Single-Value Database Fields" on page 124.

Synchronizing Garbage Strings

Question: Why is the driver synchronizing garbage strings?

Answer: The database and the third-party driver are probably using incompatible character encoding. Adjust the character encoding that your third-party driver uses.

For more information, refer to Character Encoding Values (http://java.sun.com/j2se/1.5.0/docs/guide/intl/encoding.doc.html), defined by Sun.

Running Multiple JDBC Driver Instances

Question: How do I run multiple JDBC driver instances in the same driver set? The instances require different versions of the same third-party JBDC driver (for example, the Oracle JDBC driver or the IBM DB2 Type 3 JDBC driver).

Answer: Use the Remote Loader to load each JDBC driver instance in a separate Java Virtual Machine (JVM). When run locally in the same JVM, different versions of the same third-party classes collide.

E Supported Data Types

The JDBC driver can synchronize all JDBC 1 data types and a small subset of JDBC 2 data types. How JDBC data types map to a database's native data types depends on the third-party driver.

The following list includes the supported JDBC 1 java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html).

Numeric Types:

- `java.sql.Types.BIGINT`
- `java.sql.Types.BIT`
- `java.sql.Types.DECIMAL`
- `java.sql.Types.DOUBLE`
- `java.sql.Types.NUMERIC`
- `java.sql.Types.REAL`
- `java.sql.Types.FLOAT`
- `java.sql.Types.INTEGER`
- `java.sql.Types.SMALLINT`
- `java.sql.Types.TINYINT`

String Types:

- `java.sql.Types.CHAR`
- `java.sql.Types.LONGCHAR`
- `java.sql.Types.VARCHAR`

Time Types:

- `java.sql.Types.DATE`
- `java.sql.Types.TIME`
- `java.sql.Types.TIMESTAMP`

Binary Types:

- `java.sql.Types.BINARY`
- `java.sql.Types.VARBINARY`
- `java.sql.Types.LONGVARBINARY`

The following list includes the supported JDBC 2 java.sql.Types (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Types.html).

Large Object (LOB) Types:

- `java.sql.Types.CLOB`
- `java.sql.Types.BLOB`

F java.sql.DatabaseMetaData Methods

This section lists the required and optional java.sql.DatabaseMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html) methods.

The following JDBC 1 methods are required only if the Synchronization Filter parameter is set to something other than **Exclude all tables/views**:

- getColumns(java.lang.String catalog, java.lang.String schemaPattern, java.lang.String tableNamePattern, java.lang.String columnNamePattern):java.sql.ResultSet
- getPrimaryKeys(java.lang.String catalog, java.lang.String schema, java.lang.String table):java.sql.ResultSet
- getTables(java.lang.String catalog, java.lang.String schemaPattern, java.lang.String tableNamePattern, java.lang.String[] types):java.sql.ResultSet
- storesLowerCaseIdentifiers():boolean
- storesMixedCaseIdentifiers():boolean
- storesUpperCaseIdentifiers():boolean

Optional JDBC 1 methods:

- dataDefinitionCausesTransactionCommit():boolean
- dataDefinitionIgnoredInTransactions():boolean
- getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern):java.sql.ResultSet
- getDatabaseProductName():java.lang.String
- getDatabaseProductVersion():java.lang.String
- getDriverMajorVersion():int
- getDriverMinorVersion():int
- getDriverName():java.lang.String
- getDriverVersion():java.lang.String
- getExportedKeys(java.lang.String catalog, java.lang.String schema, java.lang.String table):java.sql.ResultSet
- getMaxStatements():int
- getMaxConnections():int
- getMaxColumnsInSelect():int
- getProcedureColumns(String catalog, String schemaPattern, String procedureNamePattern, String columnNamePattern):java.sql.ResultSet
- getSchemas():java.sql.ResultSet
- getTableTypes():java.sql.ResultSet
- getUserName():java.lang.String

- supportsColumnAliasing():bolean
- supportsDataDefinitionAndDataManiuplationTransactions():boolean
- supportsDataManipulationTransactionsOnly():boolean
- supportsLimitedOuterJoins():boolean
- supportsMultipleTransactions():boolean
- supportsSchemasInDataManipulation():boolean
- supportsSchemasInProcedureCalls():boolean
- supportsTransactionIsolationLevel(int level):boolean
- supportsTransactions():boolean

Optional JDBC 2 methods:

- supportsBatchUpdates():boolean

Optional JDBC 3 methods:

- supportsGetGeneratedKeys():boolean

G JDBC Interface Methods

This section lists the JDBC interface methods (other than java.sql.DatabaseMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DatabaseMetaData.html) methods) that the JDBC driver uses. Methods are organized by class.

Often, third-party JDBC driver vendors list defects or known issues by method. You can use the following methods in collaboration with third-party JDBC driver documentation to troubleshoot or anticipate potential interoperability problems.

- java.sql.DriverManager (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html)
- java.sql.CallableStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html)
- java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html)
- java.sql.PreparedStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html)
- java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html)
- java.sql.ResultSetMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html)
- java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html)
- java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html)

The following table lists java.sql.DriverManager (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/DriverManager.html) methods that the JDBC driver uses:

Table G-1 *java.sql.DriverManager Methods*

Method Signature	JDBC Version	Required?
getConnection(String url, java.util.Properties info):java.sql.Connection	1	yes[1]
getConnection(String url, java.util.Properties info):java.sql.Connection	1	yes[1]
setLogStream(java.io.PrintStream out):void	1	no

[1]Use one method or the other.

The following table lists java.sql.CallableStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/CallableStatement.html) methods that the JDBC driver uses:

Table G-2 java.sql.CallableStatement Methods

Method Signature	JDBC Version	Required?
getBigDecimal(int parameterIndex, int scale):java.math.BigDecimal	1	yes
getBoolean(int parameterIndex):boolean	1	yes
getBoolean(String parameterName):boolean	3	no
getByte(int parameterIndex):byte	1	yes
getByte(String parameterName):byte	3	no
getBytes(int parameterIndex):byte[]	1	yes
getBytes(String parameterName):byte[]	3	no
getDate(int parameterIndex):java.sql.Date	1	yes
getDate(String parameterName):java.sql.Date	3	no
getDouble(int parameterIndex):double	1	yes
getDouble(String parameterName):double	3	no
getFloat(int parameterIndex):float	1	yes
getFloat(String parameterName):float	3	no
getInt(int parameterIndex):int	1	yes
int getInt(String parameterName)	3	no
getLong(int parameterIndex):long	1	yes
getLong(String parameterName):long	3	no
getShort(int parameterIndex):short	1	yes
getShort(String parameterName):short	3	no
getString(int parameterIndex):String	1	yes
getString(String parameterName):String	3	no
getTime(int parameterIndex):java.sql.Time	1	yes
getTime(String parameterName):java.sql.Time	3	no
getTimestamp(int parameterIndex):java.sql.Timestamp	1	yes
getTimestamp(String parameterName):java.sql.Timestamp	3	no
registerOutParameter(int parameterIndex, int sqlType):void	1	yes
wasNull():boolean	1	yes

The following table lists java.sql.Connection (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Connection.html) methods that the JDBC driver uses:

Table G-3 java.sql.Connection Methods

Method Signature	JDBC Version	Required?
close():void	1	yes
commit():void	1	no
createStatement():java.sql.Statement	1	yes
getAutoCommit():boolean	1	no
getMetaData():java.sql.DatabaseMetaData	1	yes
getTransactionIsolation():int	1	no
getWarnings():java.sql.SQLWarning	1	no
isClosed():boolean	1	no
prepareCall(String sql):java.sql.CallableStatement	1	no
prepareStatement(String sql):java.sql.PreparedStatement	1	yes
rollback():void	1	no
setAutoCommit(boolean autoCommit):void	1	no
setTransactionIsolation(int level):void	1	no

The following table lists java.sql.PreparedStatement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/PreparedStatement.html) methods that the JDBC driver uses:

Table G-4 java.sql.PreparedStatement Methods

Method Signature	JDBC Version	Required?
clearParameters() :void	1	no
execute():boolean	1	yes
executeQuery():java.sql.ResultSet	1	yes
executeUpdate():int	1	yes
setBigDecimal(int parameterIndex, java.math.BigDecimal x):void	1	yes
setBoolean(int parameterIndex, boolean x):void	1	yes
setByte(int parameterIndex, byte x):void	1	yes
setBytes(int parameterIndex, byte x[]):void	1	yes
setDate(int parameterIndex, java.sql.Date x):void	1	yes
setDouble(int parameterIndex, double x):void	1	yes
setFloat(int parameterIndex, float x):void	1	yes

Method Signature	JDBC Version	Required?
setInt(int parameterIndex, int x):void	1	yes
setLong(int parameterIndex, long x):void	1	yes
setNull(int parameterIndex, int sqlType):void	1	yes
setShort(int parameterIndex, short x):void	1	yes
setString(int parameterIndex, String x):void	1	yes
setTime(int parameterIndex, java.sql.Time x):void	1	yes
setTimestamp(int parameterIndex, java.sql.Timestamp x):void	1	yes

The following table lists java.sql.ResultSet (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html) methods that the JDBC driver uses:

Table G-5 *java.sql.ResultSet Methods*

Method Signature	JDBC Version	Required?
close():void	1	yes
getBigDecimal(int columnIndex, int scale):java.math.BigDecimal	1	yes
getBigDecimal(String columnName, int scale):java.math.BigDecimal	1	yes
getBinaryStream(int columnIndex):java.io.InputStream	1	yes
getBinaryStream(String columnName)java.io.InputStream	1	yes
getBoolean(int columnIndex):boolean	1	yes
getBoolean(String columnName):boolean	1	yes
getByte(int columnIndex):byte	1	yes
getByte(String columnName):byte	1	yes
getBytes(int columnIndex):byte[]	1	yes
getBytes(String columnName):byte[]	1	yes
getDate(int columnIndex):java.sql.Date	1	yes
getDate(String columnName)java.sql.Date	1	yes
getFloat(int columnIndex):float	1	yes
getFloat(String columnName):float	1	yes
getInt(int columnIndex):int	1	yes
getInt(String columnName):int	1	yes
getLong(int columnIndex):long	1	yes
getLong(String columnName):long	1	yes

Method Signature	JDBC Version	Required?
getMetaData():java.sql.ResultSetMetaData	1	no
getShort(int columnIndex):short	1	yes
getShort(String columnName):short	1	yes
getString(int columnIndex):String	1	yes
getString(String columnName):String	1	yes
getTime(int columnIndex):java.sql.Time	1	yes
getTime(String columnName):java.sql.Time	1	yes
getTimestamp(int columnIndex):java.sql.Timestamp	1	yes
getTimestamp(String columnName):java.sql.Timestamp	1	yes
getWarnings():java.sql.SQLWarning	1	no

The following table lists java.sql.ResultSetMetaData (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSetMetaData.html) methods that the JDBC driver uses:

Table G-6 *java.sql.ResultSetMetaData Methods*

Method Signature	JDBC Version	Required?
getColumnCount():int	1	yes
getColumnName(int column):String	1	no
getColumnType(int column):int	1	no

The following table lists java.sql.Statement (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Statement.html) methods that the JDBC driver uses:

Table G-7 *java.sql.Statement Methods*

Method Signature	JDBC Version	Required?
addBatch(java.lang.String sql):void	2	no
clearBatch():void	2	no
clearWarnings():void	1	no
close():void	1	yes
execute(java.lang.String sql):boolean	1	yes
executeBatch():int[]	2	no
executeUpdate(String sql):int	1	yes

Method Signature	JDBC Version	Required?
executeQuery(String sql):java.sql.ResultSet	1	yes
getGeneratedKeys():java.sql.ResultSet	3	no
getMoreResults():boolean	1	no
getResultSet():java.sql.ResultSet	1	yes
getUpdateCount():int	1	no
getWarnings():java.sql.SQLWarning	1	no

The following table lists java.sql.Timestamp (http://java.sun.com/j2se/1.5.0/docs/api/java/sql/Timestamp.html) methods that the JDBC driver uses:

Table G-8 *java.sql.Timestamp Methods*

Method Signature	JDBC Version	Required?
getNanos():int	1	yes
getTime():long	1	yes
setNanos(int n):void	1	yes
setTime(long time):void	1	yes
toString ():String	1	yes

H Third-Party JDBC Driver Descriptor DTD

This section contains the DTD for third-party JDBC descriptor files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT actions (exec-sql | check-for-closed-connection | fetch
metadata | rollback)*>
<!ELEMENT add-default-values-on-view-insert (#PCDATA)>
<!ELEMENT authentication (regular-expression | sql-state | error-code
| sql-state-class | error-code-range | actions)*>
<!ELEMENT check-for-closed-connection EMPTY>
<!ELEMENT column-position-comparator (#PCDATA)>\
<!ELEMENT connection-properties (property*)>
<!ELEMENT connectivity (regular-expression | sql-state | error-code |
sql-state-class | error-code-range | actions)*>
<!ELEMENT current-timestamp-stmt (#PCDATA)>
<!ELEMENT error-code (value)>
<!ATTLIST error-code
  description CDATA #IMPLIED
>
<!ELEMENT error-code-range (from, to)>
<!ATTLIST error-code-range
  description CDATA #IMPLIED
>
<!ELEMENT errors (connectivity | authentication | retry | fatal)*>
<!ELEMENT exclude-table-filter (#PCDATA)>
<!ELEMENT exec-sql (#PCDATA)>
<!ELEMENT fatal (regular-expression | sql-state | error-code | sql
state-class | error-code-range | actions)*>
<!ELEMENT fetch-metadata EMPTY>
<!ELEMENT from (#PCDATA)>
<!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)>
<!ELEMENT identity (name?, target-database?, jdbc-type?, jdbc-class?)>
<!ELEMENT import (#PCDATA)>
<!ELEMENT imports (import*)>
<!ELEMENT include-table-filter (#PCDATA)>
<!ELEMENT jdbc-class (#PCDATA)>
<!ELEMENT jdbc-driver (imports?, identity, (metadata-override |
connection-properties | sql-type-map | options | errors)*)>
<!ELEMENT jdbc-type (#PCDATA)>
<!ELEMENT key (#PCDATA)>
<!ELEMENT left-outer-join-operator (#PCDATA)>
<!ELEMENT lock-generator-class (#PCDATA)>
<!ELEMENT metadata-override (supports-schemas-in-procedure-calls?)>\
<!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT options (lock-generator-class | supports-schemas-in
```

```
metadata-retrieval | time-translator-class | column-position
comparator | use-manual-transactions | minimal-metadata | transaction
isolation-level | use-single-connection | exclude-table-filter |
include-table-filter | left-outer-join-operator | current-timestamp
stmt | add-default-values-on-view-insert | reuse-statements |
function-return-method | handle-stmt-results)*>
<!ELEMENT property (key, value)>
<!ELEMENT regular-expression (value)>
<!ELEMENT retry (regular-expression | sql-state | error-code | sql
state-class | error-code-range | actions)*>
<!ELEMENT reuse-statements (#PCDATA)>
<!ELEMENT rollback EMPTY>
<!ELEMENT sql-state (value)>
<!ATTLIST sql-state
  description CDATA #IMPLIED
>
<!ELEMENT sql-state-class (value)>
<!ATTLIST sql-state-class
  description CDATA #IMPLIED
>
<!ELEMENT sql-type-map (type*)>
<!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT supports-schemas-in-procedure-calls (#PCDATA)>
<!ELEMENT target-database (#PCDATA)>
<!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT type (from, to)>
<!ELEMENT use-manual-transactions (#PCDATA)>
<!ELEMENT use-single-connection (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

Third-Party JDBC Driver Descriptor Import DTD

This section contains the DTD for third-party JDBC descriptor import files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT actions (exec-sql | check-for-closed-connection | fetch-metadata
| rollback)*>
<!ELEMENT add-default-values-on-view-insert (#PCDATA)>
<!ELEMENT authentication (regular-expression | sql-state | error-code |
sql-state-class | error-code-range | actions)*>
<!ELEMENT check-for-closed-connection EMPTY>
<!ELEMENT column-position-comparator (#PCDATA)>
<!ELEMENT connection-properties (property*)>
<!ELEMENT connectivity (regular-expression | sql-state | error-code | sql-
state-class | error-code-range | actions)*>
<!ELEMENT current-timestamp-stmt (#PCDATA)>
<!ELEMENT error-code (value)>
<!ATTLIST error-code
  description CDATA #IMPLIED
>
<!ELEMENT error-code-range (from, to)>
<!ATTLIST error-code-range
  description CDATA #IMPLIED
>
<!ELEMENT errors (connectivity | authentication | retry | fatal)*>
<!ELEMENT exclude-table-filter (#PCDATA)>
<!ELEMENT exec-sql (#PCDATA)>
<!ELEMENT fatal (regular-expression | sql-state | error-code | sql-state-
class | error-code-range | actions)*>
<!ELEMENT fetch-metadata EMPTY>
<!ELEMENT from (#PCDATA)>
<!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)>
<!ELEMENT include-table-filter (#PCDATA)>
<!ELEMENT jdbc-driver (metadata-override | connection-properties | sql-
type-map | options | errors)*>
<!ELEMENT key (#PCDATA)>
<!ELEMENT left-outer-join-operator (#PCDATA)>\
<!ELEMENT lock-generator-class (#PCDATA)>
<!ELEMENT metadata-override (supports-schemas-in-procedure-calls?)>
<!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT options (lock-generator-class | supports-schemas-in-metadata-
retrieval | time-translator-class | column-position-comparator | use-
manual-transactions | minimal-metadata | transaction-isolation-level | use-
single-connection | exclude-table-filter | include-table-filter | left-
outer-join-operator | current-timestamp-stmt | add-default-values-on-view-
insert | reuse-statements | function-return-method | handle-stmt-results)*>
<!ELEMENT property (key, value)>
```

```
<!ELEMENT regular-expression (value)>
<!ELEMENT retry (regular-expression | sql-state | error-code | sql-state-
class | error-code-range | actions)*>
<!ELEMENT reuse-statements (#PCDATA)>
<!ELEMENT rollback EMPTY>
<!ELEMENT sql-state (value)>
<!ATTLIST sql-state
  description CDATA #IMPLIED
>
<!ELEMENT sql-state-class (value)>
<!ATTLIST sql-state-class
  description CDATA #IMPLIED
>
<!ELEMENT sql-type-map (type*)>
<!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT supports-schemas-in-procedure-calls (#PCDATA)>
<!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT type (from, to)>
<!ELEMENT use-manual-transactions (#PCDATA)>
<!ELEMENT use-single-connection (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

J Database Descriptor DTD

This section contains the DTD for database descriptor files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT add-default-values-on-view-insert (#PCDATA)>
<!ELEMENT column-position-comparator (#PCDATA)>
<!ELEMENT current-timestamp-stmt (#PCDATA)>
<!ELEMENT database (imports?, identity, options?)>
<!ELEMENT exclude-table-filter (#PCDATA)>
<!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)>
<!ELEMENT include-table-filter (#PCDATA)>
<!ELEMENT identity (name?, regex-name?, regex-version?)>
<!ELEMENT import (#PCDATA)>
<!ELEMENT imports (import*)>
<!ELEMENT left-outer-join-operator (#PCDATA)>
<!ELEMENT lock-generator-class (#PCDATA)>
<!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT options (lock-generator-class | supports-schemas-in-metadata-
retrieval | time-translator-class | column-position-comparator | use-
manual-transactions | minimal-metadata | transaction-isolation-level | use-
single-connection | exclude-table-filter | include-table-filter | left-
outer-join-operator | current-timestamp-stmt | add-default-values-on-view-
insert | reuse-statements | function-return-method | handle-stmt-results)*>
<!ELEMENT regex-name (#PCDATA)>
<!ELEMENT regex-version (#PCDATA)>
<!ELEMENT reuse-statements (#PCDATA)>\
<!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT use-manual-transactions (#PCDATA)>
<!ELEMENT use-single-connection (#PCDATA)>
```

K Database Descriptor Import DTD

This section contains the DTD for database descriptor import files.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT add-default-values-on-view-insert (#PCDATA)>
<!ELEMENT column-position-comparator (#PCDATA)>
<!ELEMENT current-timestamp-stmt (#PCDATA)>
<!ELEMENT exclude-table-filter (#PCDATA)>
<!ELEMENT function-return-method (#PCDATA)>
<!ELEMENT handle-stmt-results (#PCDATA)>
<!ELEMENT include-table-filter (#PCDATA)>
<!ELEMENT database (options?)>
<!ELEMENT left-outer-join-operator (#PCDATA)>
<!ELEMENT lock-generator-class (#PCDATA)>
<!ELEMENT minimal-metadata (#PCDATA)>
<!ELEMENT options (lock-generator-class | supports-schemas-in
metadata-retrieval | time-translator-class | column-position-comparator |
use-manual-transactions | minimal-metadata | transaction-isolation-level |
use-single-connection | exclude-table-filter | include-table-filter | left-
outer-join-operator | current-timestamp-stmt | add-default-values-on-view-
insert | reuse-statements | function-return-method | handle-stmt-results)*>
<!ELEMENT reuse-statements (#PCDATA)>
<!ELEMENT supports-schemas-in-metadata-retrieval (#PCDATA)>
<!ELEMENT time-translator-class (#PCDATA)>
<!ELEMENT transaction-isolation-level (#PCDATA)>
<!ELEMENT use-manual-transactions (#PCDATA)>
<!ELEMENT use-single-connection (#PCDATA)>
```

L Policy Example: Trigger-less Future Event Processing

The following example assumes that a “commence” attribute exists and does the following:

- Holds the time stamp value indicating when an event should be processed
- Contains an integer or Java string time stamp value. See “Time Syntax” on page 60.

```
<policy    xmlns:Timestamp="http://www.novell.com/nxsl/java/
java.sql.Timestamp"
  xmlns:TimestampUtil="http://www.novell.com/nxsl/java/
com.novell.nds.dirxml.driver.jdbc.db.TimestampUtil"
  xmlns:jdbc="urn:dirxml:jdbc">
<rule>
<description>Get commencement date from datasource.</description>
  <conditions>
    <and>
      <if-xpath op="true">.</if-xpath>
    </and>
  </conditions>
  <actions>
    <do-set-local-variable name="commence">
      <arg-string>
        <token-src-attr class-name="User" name="commence"/>
      </arg-string>
    </do-set-local-variable>
  </actions>
</rule>
```

```
<rule>
  <description>Break if commencement date unavailable.</description>
  <conditions>
    <and>
      <if-local-variable name="commence" op="equal"/>
    </and>
  </conditions>
  <actions>
    <do-break/>
  </actions>
</rule>
```

```
<rule>
<description>Parse times.</description>
  <conditions>
    <and>
      <if-xpath op="true">.</if-xpath>
    </and>
  </conditions>
  <actions>
    <do-set-local-variable name="dbTime">
      <arg-object>
        <token-xpath expression="Timestamp:valueOf(@jdbc:database-local-
time)"/>
      </arg-object>
    </do-set-local-variable>
    <do-set-local-variable name="eventTime">
      <arg-object>
        <token-xpath expression="Timestamp:valueOf($commence)"/>
      </arg-object>
    </do-set-local-variable>
  </actions>
</rule>
<rule>
  <description>Is commencement date after database time?</description>
  <conditions>
    <and>
      <if-xpath op="true">.</if-xpath>
    </and>
  </conditions>
  <actions>
    <do-set-local-variable name="after">
      <arg-string>
        <token-xpath expression="TimestampUtil:after($eventTime, $dbTime)"/
>
      </arg-string>
    </do-set-local-variable>
  </actions>
</rule>

<rule>
<description>Retry if future event.</description>
  <conditions>
    <and>
      <if-local-variable name="after" op="equal">true</if-local-variable>
    </and>
  </conditions>
  <actions>
    <do-status level="retry">
      <arg-string>
        <token-text xml:space="preserve">Future event detected.</token-
text>
      </arg-string>
    </do-status>
  </actions>
</rule>
</policy>
```

M Setting Up an OCI Client on Linux

- "Downloading the Instant Client" on page 245
- "Setting Up the OCI Client" on page 245
- "Configuring the OCI Driver" on page 246

Downloading the Instant Client

1. Download the Oracle Instant Client (`instantclient-basic-linux32-11.1.0.7.zip`).

   The file is available from Instant Client Downloads (http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html).

2. Download the Oracle SQL Plus binary (`instantclient-sqlplus-linux32-11.1.0.7.zip`).

   The file is available from Instant Client Downloads (http://www.oracle.com/technology/software/tech/oci/instantclient/htdocs/linuxsoft.html).

Setting Up the OCI Client

Set up the Oracle Instant Client on the machine where the JDBC driver is running (not on the machine where Oracle is running).

1. Log into Linux as `root`, and create the following structure:

   `/oracle` `/oracle/client` `/oracle/client/bin` `/oracle/client/lib` `/oracle/client/network/admin`

2. Unzip all files from `instantclient-basic-linux32-11.1.0.7.zip` to `/oracle/client/lib`.
3. Unzip all files from `instantclient-sqlplus-linux32-11.1.0.7.zip` to `/oracle/client/bin`.
4. Copy `libsqlplus.so` from `/oracle/client/bin` to `/oracle/client/lib`.
5. Copy `libsqlplusic.so` from `/oracle/client/bin` to `/oracle/client/lib`.
6. Using `chmod`, ensure that the file `sqlplus` in `/oracle/client/bin` is executable.
7. Copy a valid `tnsnames.ora` into `/oracle/client/network/admin`.

   If you don't have a `tnsnames.ora` file, use the Oracle configuration tool to create one.

   Make sure that the `tnsnames.ora` filename is in lowercase.

8. Modify the `profile.local` file by adding the following lines:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/oracle/client/lib

export TNS_ADMIN=/oracle/client/network/admin

export PATH=$PATH:/oracle/client/lib
```

The `profile.local` file is in the `/etc` folder. If the file doesn't exist, create one. The file can consist of only the three export lines.

The `profile.local` file extends the LD_LIBRARY_PATH, sets TNS_ADMIN, and extends the PATH. This file is read when the server boots.

9 Ensure that the exports in the `profile.local` file are always valid.

10 Copy the appropriate jar files to the Identity Manager classes directory.

These `.jar` files are supplied with the Instant Client.

The Identity Manager classes directory is the directory where your driver is located.

11 Start SQL Plus with the following example command (assuming that the directory is `/oracle/client/bin`):

```
./sqlplus username/password@sid
```

Configuring the OCI Driver

To configure the driver, customize the driver's URL syntax. See Table 14-20 on page 194.

An example URL syntax is jdbc:oracle:oci8:@ORACLE10. In this example, **ORACLE10** is the connection string in the `tnsnames.ora` file.

Figure M-1 *Example tnsnames.ora File*

N Sybase Chain Modes and the OpenText Identity Manager JDBC driver

Sybase can execute stored procedures in two distinct modes: chained and unchained. Depending upon the configuration of the Identity Manager JDBC driver and stored procedures in a database, various problems can arise. This section can help you understand and resolve those problems.

- "Error Codes" on page 247
- "Procedures and Modes" on page 248

Error Codes

- "Error 226: SET CHAINED command not allowed within multi-statement transaction" on page 247
- "Error 7112: Stored procedure 'x' may be run only in chained transaction mode" on page 247
- "Error 7113: Stored procedure 'x' may be run only in unchained transaction mode" on page 248

Error 226: SET CHAINED command not allowed within multi-statement transaction

Effect: Throws the exception of com.sybase.jdbc2.jdbc.SybSQLException with error code 226 and an SQL state of ZZZZZ.

Cause: This exception is usually caused by a defect in older versions of jConnect.

Solution: Download and upgrade to the latest version. Downloads are available at the jConnect for JDBC Web page (http://www.sybase.com/products/informationmanagement/softwaredeveloperkit/jconnect).

Error 7112: Stored procedure 'x' may be run only in chained transaction mode

Effect: Throws the exception of com.sybase.jdbc2.jdbc.SybSQLException with error code 7712 and an SQL state of ZZZZZ.

Cause: The stored procedure was created in chained mode, or later altered to run in chained mode, but the driver is currently running in unchained mode. The probable cause is that the Use Manual Transactions? parameter is set to **False**. Another possibility is that the transaction type has been overridden to **auto** in a policy.

Solution: Do one of the following:

- Use stored procedure sp_procxmode to change the stored procedure's mode to **unchained** or **anymode** (preferred).
- Change the driver's Use Manual Transactions? parameter to **True**, or change the policy transaction type to **manual**.

Error 7113: Stored procedure 'x' may be run only in unchained transaction mode

Effect: Throws the exception com.sybase.jdbc2.jdbc.SybSQLException with error code 7713 and an SQL state of ZZZZZ.

Cause: The stored procedure was created in unchained mode, or later altered to run in unchained mode, but the driver is currently running in chained mode. The probable cause is that the Use Manual Transactions? parameter is set to **True**. Another possibility is that the transaction type has been overridden to **manual** in policy.

Solution: Do one of the following:

- Use stored procedure sp_procxmode to change the stored procedure's mode to **chained** or **anymode** (preferred).
- Change the driver's Use Manual Transactions? parameter to **False**, or change the policy transaction type to **auto**.

If you set use-manual-transactions to **False**, all transactions consist of a maximum of one statement.

Procedures and Modes

- "Using Stored Procedure sp_proxmode" on page 248
- "Chained and Unchained Modes" on page 249
- "Managing Transactions in a Policy" on page 249
- "Useful Links" on page 250

Using Stored Procedure sp_proxmode

The preferred way to avoid errors 7112 and 7113 is to alter all stored procedures invoked directly or indirectly by the driver (via triggers, for example) to run in both chained and unchained mode. To alter a procedure, invoke the sp_procxmode procedure with two arguments:.

- The procedure name
- The mode

The following example illustrates how to invoke the sp_procxmode procedure from the isql command line:

```
client:sp_procxmode my_procedure, anymode go
```

Of course, not all customers are willing to alter stored procedure modes. Altering a procedure's mode might alter its runtime behavior, which could alter the behavior of other applications that invoke the procedure.

Chained and Unchained Modes

Unchained mode is Sybase's native way of executing SQL. A second mode, chained mode, was later added to make the database compatible with SQL standards.

Table N-1 *Modes and Compatibility*

Mode	Compatibility
Chained	SQL-compatible mode
Unchained	Sybase native mode

Sybase provides a third-party JDBC driver called jConnect. The default mode of jConnect is unchained. Whenever the method Connection.setAutoCommit(boolean autoCommit):void is invoked, jConnect switches modes. See java.sql Interface Connection (http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html).

Table N-2 *Methods and Switches*

Method	Effect
Connection.setAutoCommit(true)	Switches to unchained mode
Connection.setAutoCommit(false)	Switches to chained mode

If the Use Manual Transactions? parameter is set to **False**, the driver invokes Connection.setAutoCommit(true). That is, the driver enters unchained mode. This is the normal processing mode for SELECT statements and SQL embedded in a policy where the transaction type is set to `auto`. See "Manual vs. Automatic Transactions" on page 147. When the driver is in this state, any chained stored procedures invoked directly or indirectly by the driver yield the 7112 error.

If the Use Manual Transactions? parameter is set to **True**, the driver invokes Connection.setAutoCommit(false). That is, the driver enters chained mode. This is the normal processing mode for all statements except SELECT statements and SQL embedded in a policy where the transaction type is set to `manual`. See "Manual vs. Automatic Transactions" on page 147. When the driver is in this state, any unchained stored procedures invoked directly or indirectly by the driver yield the 7113 error.

Managing Transactions in a Policy

For information on managing transactions in a policy, see "Manual vs. Automatic Transactions" on page 147

Useful Links

- Transaction modes and stored procedures (http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/55096;hf=0;pt=55096#X) in the *Transact-SQL User's Guide*
- Selecting the transaction mode and isolation level (http://manuals.sybase.com/onlinebooks/group-as/asg1250e/sqlug/@Generic__BookTextView/53713;pt=53001) in the *Transact-SQL User's Guide*

O Driver Properties

This section provides information about the Driver Configuration and Global Configuration Values properties for the JDBC driver and the Fan-out driver. These are the only unique properties for drivers. All other driver properties (Named Password, Engine Control Values, Log Level, and so forth) are common to all drivers. Refer to "Driver Properties (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/b94pq23.html)" in the *OpenText™ Identity Manager CE 24.4 (v4.10) Driver Administration Guide (https://www.netiq.com/documentation/identity-manager-4.10/driver_admin/data/bktitle.html)* for information about the common properties.

The information is presented from Identity Console's perspective. If a field is different in Designer, it is marked with a Designer icon.

- "Driver Configuration" on page 251
- "Global Configuration Values" on page 255

Driver Configuration

In Identity Console:

1. Click the **IDM Administration** tile.
2. On the Driver Dashboard, locate the driver, then click the driver icon to display the driver's properties page.

In Designer:

1. Open a project in the Modeler.
2. Right-click the driver icon or line, then select click **Properties > Driver Configuration.**

The following sections describe driver configuration in details:

- "Driver Module" on page 251
- "Authentication" on page 252
- "Startup Option" on page 252
- "Driver Parameters" on page 253
- "ECMAScript" on page 255
- "Global Configuration" on page 255

Driver Module

The driver module changes the driver from running locally to running remotely or the reverse.

Java: Use this option to specify the name of the Java class that is instantiated for the shim component of the driver. This class can be located in the `classes` directory as a class file, or in the `lib` directory as a `.jar` file. If this option is selected, the driver is running locally. Select this option to run the driver locally.

The Java class name is: `com.netiq.idm.driver.fanoutshim.FanoutDriverShim`.

Native: This option is not used with the REST driver.

Connect to Remote Loader: This option is not used for JDBC Fan-out driver. Used when the driver is connecting remotely to the connected system.

Name: Displays the Java class name.

Driver Object Password: Use this option to set a password for the driver object. If you are using the Fan-out Agent, you must enter a password on this page. This password is used by the Fan-out Agent to authenticate itself to the driver shim.

Authentication

The authentication section describes the parameters required for authentication to the connected database.

Authentication ID: Specify a user application ID. This ID is used to pass Identity Vault subscription information to the application. For example, `Administrator`.

Connection Information (Designer only): Specify the IP address or name of the server the application shim should communicate with.

IMPORTANT: The Remote Loader options are not applicable for the JDBC Fan-out drivers. The Fan-out drivers use the Fan-out Agent component to create multiple database instances.

Driver Cache Limit (kilobytes): Specify the maximum event cache file size (in KB). If it is set to zero, the file size is unlimited. select **Unlimited** option to set the file size to unlimited in Designer.

Application Password: Use the **Set Password** option to set the application authentication password.

Remote loader password: Use this option to update the remote loader password. This option is not used for the JDBC Fan-out driver.

Startup Option

The Startup Option section allows you to set the driver state when the Identity Manager server is started.

Auto start: The driver starts every time the Identity Manager server is started.

Manual: The driver does not start when the Identity Manager server is started. The driver must be started through Designer or Identity Console.

Disabled: The driver has a cache file that stores all of the events. When the driver is set to Disabled, this file is deleted and no new events are stored in the file until the driver state is changed to Manual or Auto Start.

Driver Parameters

The Driver Parameters section lets you configure the driver-specific parameters. When you change driver parameters, you tune driver behavior to align with your network environment.

The parameters are presented by category:

- "Driver Settings" on page 253
- "Normal JDBC Driver Settings" on page 254
- "Subscriber Settings" on page 254
- "Publisher Settings" on page 255

Driver Settings

Fanout transport related parameters: Select **Show** to view the transport related parameters for Fan-out drivers.

Show Subscriber Event Queue parameters: Select **Show** to view the subscriber event parameters. The options are:

- SEND: The queue for sending the subscriber events to the Fan-out Agent.
- RECV: The subscriber event receiving queue for receiving the subscriber events from Fan-out Agent.
- DELAYED RECV: The subscriber delayed event receiving queue is used for receiving the delayed subscriber events from FanOut Agent.

Show Configuration Queue Parameters: Select **Show** to view the configuration queue parameters. The options are SEND and RECV.

Show Query-in Queue Parameters: Select **Show** to view the query-in queue parameters. The options are SEND and RECV.

Show Query-out Queue Parameters: Select **Show** to view the query-out queue parameters. The options are SEND and RECV.

Show Other Parameters: Select **Show** to view the additional parameters.

- Configuration batch size: Specify the batch size for the Driver configuration document. The value is from 1 - 99999.

Show Fanout Parameters: Select **Show** to view the fan-out connection related information such as Fan-out Agent password, configuration information, Fan-out Agent shim password.

- Fanout Shim Password: Specify the password for the fan-out driver shim. After successful authentication, the FanOut Agent loads/creates the driver instances of the specified shim class name.
- Fanout Agent Password: Specify the password of Fan-out Agent you are connecting to. The Fan-out Agent establishes connection only after a valid authentication.
- Encryption Key: Specify the key to encrypt/decrypt the sensitive data before sending to the message queue(s).
- AMQ Keystore Key: Specify the full path to the keystore file.

- AMQ Keystore Password: Specify the keystore password.
- AMQ Truststore Path for SSL Certs: Specify the full path to the truststore file.
- AMQ Truststore Password: Specify the truststore.
- Fanout Shim classname: Specify shim classname that the Fan-out Agent loads when you start the any fan-out driver.
- Matching Attributes: Matching attributes that Fan-out Agent uses to match objects in delayed add events. Attribute names must be as per the schema of the connected system. NetIq recommends that these attributes must be schema-mapped equivalent of the attributes that are used in the object matching policy.

Normal JDBC Driver Settings

For the normal JDBC driver setting, see Driver Parameters.

Subscriber Settings

Disable Subscriber: Select no (default) to allow flow of events from Identity Manager engine to the connected database.

Show primary key parameters: Select **Show** if you want to configure the primary key parameters.

- **Generation/retrieval method (table-global):** Select the desired option to generate/retrieve the primary key values. This setting is global for all tables and views. The options are as follows:
  - subscription event (default)
  - subscriber-generated
  - auto-generated / identity column
- **Retrieval timing (table-global):** Select the desired option to retrieve the primary key value. This setting is global for all tables and views. The options are:
  - before row insertion (default)
  - after row insertion
- **Method and timing (table-global):** Specify how and when the primary key values are generated or retrieved on a per table or view basis. This parameter overrides global method and timing settings. Use semicolon, comma, or space as the delimiter for multiple values. For example: `usr("?=indirect.proc_idu()"); grp("indirect.proc_idg(idg)")`.

Disable statement-level locking: Select the appropriate option to disable statement locking. This option determines if explicit locking or database resources are disabled on the Subscriber channel. The value is set to no (default) by default.

Check update counts: Select yes (default) to enable the Subscriber channel to check for any updates after any of the insert, update, or delete statements are executed against the tables. This option ensures that the statements are resulting in updating the database. The value is set to yes (default) by default.

Publisher Settings

Disable the Publisher Channel: Select **yes (default)** to ignore the flow of events from the connected database to Identity Manager engine. The Fan-out driver implementation do not support the Publisher channel. By default, this option is disabled for the Fan-out driver configuration.

Heartbeat interval (in minutes): Specify the interval in minutes that the Publisher remain inactive before sending a heartbeat document.

ECMAScript

Displays an ordered list of ECMAScript resource files. The files contain extension functions for the driver that Identity Manager loads when the driver starts. You can add additional files, remove existing files, or change the order the files are executed.

Global Configuration

Displays an ordered list of Global Configuration objects. The objects contain extension GCV definitions for the driver that Identity Manager loads when the driver is started. You can add or remove the Global Configuration objects, and you can change the order in which the objects are executed.

Global Configuration Values

Global configuration values (GCVs) are values that can be used by the driver to control functionality. GCVs are defined on the driver or on the driver set. Driver set GCVs can be used by all drivers in the driver set. Driver GCVs can be used only by the driver on which they are defined.

The REST driver includes several predefined GCVs. You can also add your own if you discover you need additional ones as you implement policies in the driver.

To access the driver's GCVs in Identity Console:

1. Click the **IDM Administration** tile.
2. On the Driver Dashboard, locate the driver, then click the driver icon to display the driver's properties page.
   - **2a** Select the **Configuration** tab.
   - **2b** Expand the **Global Config Values** section.

To add a GCV to the driver set:

1. On the Driver Dashboard, click the upper right corner of the driver set to display the Action menu.
2. Select **Driver Set Properties**.
3. On the **Driver Set Configuration** tab, expand the **Global Config Values** section.
4. Save the values.

To access the driver's GCVs in Designer:

1 Open a project in the Modeler.

2 Right-click the driver icon or line, then select **Properties > Global Configuration Values.**

or

To add a GCV to the driver set, right-clickthe driver set icon , then click **Properties > GCVs**.

The global configuration values are organized as follows:

Global Configuration Values

The following global configuration values are used for database options and base configuration options.

JDBC connection URL format used: Specify the connection URL format used for the JDBC driver to connect to the databases. Use '<HOST>','<PORT> and '<DB>' tokens to specify the location of host's IP address, port and database/SID in the connection URL.

NOTE: The tokens are case-sensitive and angle-brackets are mandatory since they are used as delimiters.

If you use the same fan-out driver to connect oracle pluggable database and oracle traditional database, the url template of the databases should be separated using a comma. For example: jdbc:oracle:thin:@<HOST>:<PORT>/<DB>, jdbc:oracle:thin:@<HOST>:<PORT>:<DB>

Synchronization model: Select the synchronization model. The synchronization options are: Direct and Indirect. Direct synchronization uses views to synchronize directly to existing tables of arbitrary structure. Indirect synchronization synchronizes to intermediate staging tables with a particular structure.

UserName Column: Specify the exact column name of the **usr** table that store the usernames.

Managed System Information

These settings help Identity Reporting to generate reports. There are different sections in the **Managed System Information** tab.

- "General Information" on page 256
- "System Ownership" on page 257
- "System Classification" on page 257
- "Connection and Miscellaneous Information" on page 257
- "JDBC FanOut Instances Information" on page 257

General Information

Name: Specify a descriptive name for the managed system.

Description: Specify a brief description of the managed system.

Location: Specify the physical location of the managed system.

Vendor: Specify Microsoft as the vendor of the managed system.

Version: Specify the version of the managed system.

System Ownership

Business Owner: Browse to and select the business owner in the Identity Vault for the connected application. You must select a user object, not a role, group, or container.

Application Owner: Browse to and select the application owner in the Identity Vault for the connected application. You must select a user object, not a role, group, or container.

System Classification

Classification: Select the classification of the connected application. This information is displayed in the reports. The options are:

- Mission-Critical
- Vital
- Not-Critical
- Other

  If you select **Other**, you must specify a custom classification for the connected application.

Environment: Select the type of environment the connected application provides. The options are:

- Development
- Test
- Staging
- Production
- Other

  If you select **Other**, you must specify a custom classification for the connected application.

Connection and Miscellaneous Information

Connection and miscellaneous information: This set of options is always set to **hide**, so that you don't make changes to these options. These options are system options that are necessary for reporting to work.

JDBC FanOut Instances Information

These settings help to configure the Managed System Service related details of each JDBC FanOut instance. To create a new instance, click the plus sign and fill in the following information:

- **JDBC FanOut Instance Name:** Specify the descriptive name of the new logical instance of the managed system.
- **Show other configuration values:** Select **Show** to display additional information related to the FanOut instance. For more information, see "Managed System Information" on page 256.

- **Connection and miscellaneous information:** Select **Show** to display the system options. The options are:
  - Instance ID
  - Authentication IP Address
  - Authentication Port
  - Authentication ID
  - Database Schema
  - Type

NOTE: The connection information options are auto-generated and always set to **hide**.

Entitlements

There are multiple sections in the **Entitlements** tab. Depending on which packages you installed, different options are enabled or displayed.

- "Entitlements" on page 258
- "Data Collection" on page 258
- "Role Mapping" on page 259
- "Resource Mapping" on page 259
- "Parameter Format" on page 259
- "Entitlements Extensions" on page 259

Entitlements

Account Entitlement Value: Specify the entitlement value to assign for user account during the account creation. Identity Applications display this value to the user during account provisioning.

Use Entitlements to Control DB Accounts: Select **True** to enable the driver to manage database accounts based on the driver's defined entitlements. Select **False** to disable management of database accounts based on the entitlements.

Use Group Entitlement: Select **True** to enable the driver to manage group membership based on the driver's defined entitlements.

Allow Login Disabled in Subscriber Channel: Select **True** to enable the driver to control the flow of **Login Disabled** attribute in the Subscriber Channel and only on a regular attribute change.

Advanced Settings: Entitlement options that allow or deny additional functionality like data collection, role mapping, resource mapping, parameter format, and entitlement extensions. Leave these settings as default.

Data Collection

Data collection enables Identity Report to gather information to generate reports. For more information, see the *Administrator Guide to Identity Reporting (https://www.netiq.com/documentation/identity-manager-4.10/report_setup/data/bookinfo.html)*.

Enable data collection: Select **Yes** to enable data collection for the driver through Data Collection Service by the Managed System Gateway driver. If you are not going to run reports on data collected by this driver, select **No**.

Allow data collection from user accounts: Select **Yes** to allow data collection by Data Collection Service for the user accounts.

Allow data collection from groups: Select **Yes** to allow data collection by Data Collection Service for groups.

Role Mapping

The Identity Manager Identity Applications allows you to map business roles with IT roles. For more information, see *Identity Applications Administration* in the *OpenText™ Identity Manager - Administrator's Guide to the Identity Applications (https://www.netiq.com/documentation/identity-manager-4.10/identity_apps_admin/data/bookinfo.html)*.

Enable role mapping: Select **Yes** to make this driver visible to Identity Applications.

Allow mapping of user accounts: Select **Yes** if you want to allow mapping of user accounts in Identity Applications. An account is required before a role, profile, or license can be granted through Identity Applications.

Allow mapping of groups: Select **Yes** if you want to allow mapping of groups in Identity Applications.

Resource Mapping

Identity Applications allow you to map resources to users. For more information, see the *OpenText™ Identity Manager CE 24.4 (v4.10) - User's Guide to the Identity Applications (https://www.netiq.com/documentation/identity-manager-4.10/identity_apps_user/data/netiq-identity-manager-user-guide.html)*.

Enables resource mapping: Select **Yes** to make this driver visible to Identity Applications.

Allow mapping of user accounts: Select Yes if you want to allow mapping of user accounts in Identity Applications. An account is required before a role, profile, or license can be granted.

Allow mapping of groups: Select **Yes** if you want to allow mapping of groups in Identity Applications.

Parameter Format

Format for Account entitlement: Specify the parameter format the entitlement agent must use when granting the user account entitlement. The options are **Identity Manager 4** and **Legacy**.

Format for Group entitlement: Specify the parameter format the entitlement agent must use when granting the group entitlement. The options are **Identity Manager 4** and **Legacy**.

Entitlements Extensions

User account extension: Specify the user account extension. The content of this field is added below the entitlement elements in the EntitlementConfiguration resource object

Group extensions: Specify the group extensions. The content of this field is added below the entitlement elements in the EntitlementConfiguration resource object

Account Tracking

The following controls the Account tracking is part of Identity Reporting. For more information, see the *Administrator Guide to Identity Reporting (https://www.netiq.com/documentation/identity-manager-4.10/report_setup/data/bookinfo.html)*.

Enable Account Tracking: Set this to **True** to enable account tracking policies for the JDBC Fan-out driver. Set it to **False** if you do not want to execute account tracking policies.

- Object class
- Realm
- Identifiers for Account
- Status Attribute
- Status active value
- Status inactive value
- Subscription default status
- Publication default status

Password Synchronization

The following GCVs control password synchronization for the Office 365 driver. For more information, see the *OpenText™ Identity Manager CE 24.4 (v4.10) Password Management Guide (https://www.netiq.com/documentation/identity-manager-4.10/password_management/data/netiq-idm-password-management.html)*.

Application Accepts Passwords from Identity Manager: If this option is set to **True**, the driver allows passwords to flow from the Identity Manager data store to the connected Office 365 server.

Identity Manager Accepts Passwords from the Application: If this option is set to **True**, it allows passwords to flow from the connected system to Identity Manager.

Publish Passwords to NDS Password: Use the password from the connected system to set the non-reversible NDS password in the Identity Vault.

Publish Passwords to Distribution Password: Use the password from the connected system to set the NMAS Distribution Password used for Identity Manager password synchronization.

Require passwords policy validation before publishing passwords: Select **True** to apply NMAS password policies when publishing passwords. Password is not written to the data store if it does not comply.

Reset user's external system password to the Identity Manager password on failure: If this option is set to **True**, and the Distribution Password fails to distribute, attempt to reset the password in the connected system by using the Distribution Password from the Identity Manager data store.

Notify the user of password synchronization failure via e-mail: If this option is set to **True**, notify the user by e-mail of any password synchronization failures.

Connected System or Driver Name: Specifies the name of the connected system, application or Identity Manager driver. This value is used by the e-mail notification templates to identify the source of notification messages.

In Designer, you must click the icon next to a GCV to edit it. This displays the Password Synchronization Options dialog box for a better view of the relationship between the different GCVs.

In Identity Console, to edit the Password management option, follow the steps given below:

1. Click the **IDM Administration** tile.
2. On the Driver Dashboard, locate the driver, then click the driver icon to display the driver's properties page.
3. Select the **Configuration** tab.
4. Expand the **Global Config Values** section.
5. Select the **Password Synchronization** tab.

JDBC Fanout Common

Allow 'Group add' in Fanout mode: Select to Disable to prevent the group add/creation events in the Subscriber channel. Group add events are broadcasted to each of the instances configured by the driver. If disabled, group add operations will be vetoed.

Synchronize the first or the last replica value: Select the appropriate option to synchronize the first or last replica value of multi-valued attributes mapped to single-valued columns. The options are: First and Last.